Naval Research Laboratory

Washington, DC 20375-5320

Adding Semantic Support to Existing UDDI Infrastructure

JIM LUO
BRUCE MONTROSE
MYONG KANG

Center for High Assurance Computer Systems
Information Technology Division

October 31, 2005

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
31-10-2005	Memorandum Report	

4. TITLE AND SUBTITLE

Adding Semantic Support to Existing UDDI Infrastructure

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Jim Luo, Bruce Montrose, and Myong Kang

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
55-8089-W-6

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory, Code 5542
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5540--05-8918

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

10. SPONSOR / MONITOR'S ACRONYM(S)

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

UDDI is not capable of handling semantic markups for Web services due to its flat data model and limited search capabilities. In this paper, we introduce an approach to support semantic service description and match making with registries that conforms to UDDO V3 specification. Specifically, we discuss how to store complex semantic markups in UDDI data model and use that information to perform semantic query processing. Our approach does not require any modification to the existing UDDI registries. The add-on modules reside only on clients who wish to take advantage of semantic capabilities. This approach is completely backward compatible and can integrate seamlessly into existing UDDI infrastructure.

15. SUBJECT TERMS

Semantic web; Service-oriented architecture; Semantic registry

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	UL	58	Jim Luo
Unclassified	Unclassified	Unclassified			**19b. TELEPHONE NUMBER** *(include area code)* (202) 404-8764

Table of Content

Table of Figures

Adding Semantic Support to Existing UDDI Infrastructure

1. Introduction

Automatic discovery of Web services is an important capability for Service-Oriented Architecture (SOA). The first step in providing this capability is to markup Web services with metadata in a well-understood and consistent manner. The W3C community developed the Web Ontology Language (OWL) to address this problem [1]. It is a machine-understandable description language that is capable of describing resources in a manner much richer than traditional flat taxonomies and classification systems. OWL-S is an OWL-based ontology that provides a core set of concepts specifically for describing Web services [2]. It allows for descriptions of Web service inputs, outputs, operations, and categorization using OWL ontology concepts.

The next step in providing automatic discovery is to advertise those service descriptions in a registry capable of fine-grained semantic matchmaking. Universal Description, Discovery and Integration (UDDI) is a Web-based distributed registry standard for the SOA [3]. It is one of the central elements of the interoperable framework and an OASIS standard with major backers including IBM and Microsoft. The specification is becoming widely accepted as a Web infrastructure standard and is already seeing widespread use in companies, government agencies, and the military. However, UDDI is not capable of storing and processing semantic service descriptions written in OWL.

It is clear that a registry that supports semantic annotation and matchmaking for Web services will produce much more refined search results [4, 5]. However, the existing registry standard does not support semantics. There are several possible approaches in developing a semantic registry. The first approach is to completely abandon the UDDI specification and create a new registry standard specifically designed for OWL with a new data model, API, and implementations. It would cost a great and all previous investments in UDDI infrastructure will become worthless. Implementation of this approach is not realistic without a critical mass of organizations already using OWL based semantic service descriptions. The second possible approach is to introduce small modifications to the UDDI specification to provide for semantic support. The authors of the specification made a concerted effort not to limit the potential capability of the conforming registries. They provide a framework for extensions so that registry implementations can provide additional capabilities not included in the original UDDI specification. While the specification supports extensions, this approach will still cost a great deal requiring significant modifications to the registry implementations and existing infrastructure. The third possibility is to incorporate OWL into existing UDDI registry implementations without modification. This could involve dropping support for certain aspects of OWL functionality, but it would seamlessly integrate into the existing Web services infrastructure and require very little additional cost.

We chose to pursue the third approach by interoperating OWL with existing registry implementations that conform to the UDDI version 3 specification [6]. The algorithm and prototype of the approach are presented in this paper. Even though semantic matchmaking can produce much more refined results than the UDDI style syntax queries, it is not yet accepted by mainstream users. The ability to use existing registries would allow organizations to

experiment with semantic registries while protecting their existing investments in SOA infrastructure. The UDDI registries are left intact and modifications are only necessary on the client-side. Semantic support can be seamlessly integrated into existing registry operations.

Storing and processing OWL-based semantic service descriptions in existing UDDI registries may not be the ideal solution in the long run. It will have drawbacks with respect to functionality as well as efficiency in terms of both speed and storage compared to registries specifically designed to handle semantic service description and queries. However, this approach will provide a very cost effective and functional short-term solution that builds on top of existing registry infrastructure. Even though our prototype cannot fully support all aspects of the OWL language, functionalities commonly used for description and matchmaking of Web services are supported. Furthermore, we hope to influence future development of the UDDI standard to include features that will facilitate support for OWL so that the two standards can converge and fully interoperate.

The rest of this paper is organized as follows. A few convincing reasons for semantic markups of Web services are presented in Section 2. Section 3 presents a high-level overview of our approach. Section 4 lists the types of queries our prototype can support. Section 5 gives specifics of the exact translation between OWL and the UDDI data model. Section 6 describes the matchmaker in detail. Recommendations for the next version of the UDDI standard to facilitate semantic support are presented in section 7. Section 8 presents related work in semantic registries. A conclusion is made in section 9.

2. The Capabilities of Semantic Markup of Web Services

Currently, the Service Oriented Architecture relies on syntax based standards such as UDDI and WSDL [7]. However, as service descriptions and requests become more complicated, automatic and effective matchmaking will require semantic metadata. OWL and OWL-S address this problem by allowing for creation of ontologies that can be used to add semantic meaning to concepts in a particular domain of interest. OWL adds the following capabilities in the context of service description markup and matchmaking.

Dynamic Discovery
Existing syntax based SOA standards such as WSDL allow for dynamic invocation of Web services. However, this is only true for Web services that have already been discovered. Dynamic discovery, on the other hand, needs to deal with unknown services. It will require the use of semantic metadata that software agents can understand and interpret without human intervention. WSDL does not allow attachment of semantic meaning to data. Identical syntactic data can have entirely different semantic meanings. For example, the WSDL parameters specifications for a BookBuyer service and a BookSeller service can be identical. They can both have "book name" as input and "price" as output. However, the same price output refers to the buying price for BookBuyer and the selling price for the BookSeller. OWL-S can be used to add semantics to inputs and outputs of the Web service by attaching semantic metadata that reference ontology concepts. Agents can then understand service

descriptions without intervention from the human user thus enabling dynamic discovery as well as dynamic invocation of Web services.

Class Hierarchy

Ontologies organize concepts into class and property hierarchies. This allows for specification of service descriptions and requests in various levels of detail. Matchmakers that understand the class and property hierarchies will be able to match related concepts even if they are syntactically distinct.

Property Definition

Ontologies allow for refinement of concepts by attachment of properties. Using properties, composite concepts can be defined that do not exist in the original ontology. This allows for fine-grained definition of concepts in various levels of detail that can be used in service descriptions.

Inference

Ontologies establish relationships between concepts. Inference is the deduction process based on those relationships established in the ontology. Statements not explicitly stated in the ontology can be inferred using ontology reasoners. For example, if a service is described with support for encryption, ontology aware matchmakers can reason that the service is secure. Queries for secure services will yield the service description even though it is not explicitly defined as a secure service.

3. System Overview

OWL allows for semantic description of Web services based on shared ontologies. The ontologies are developed by industry in a particular domain or by standards bodies. Individual organizations then publish service descriptions based on classes established in the ontologies. This is analogous to the systems of categorization and identification envisioned by the current UDDI specification. However, the information contained in ontologies is much richer and their use is much more complex.

There are two major differences that make OWL incompatible with UDDI in its current state.
- Ontology data are much richer and complex than the flat categorization and identification systems used by UDDI. OWL supports *property annotations* that allow service descriptions to assert more specific facts about the members of classes or instances. OWL service descriptions use properties to create *composite concepts* that contain multiple layers of annotations. The UDDI data model, however, is not capable of storing service descriptions involving composite concepts
- Ontologies establish semantic relationships among concepts. Therefore, matching of ontology concepts must be done semantically. The class and property hierarchies must be taken into account by the query engine when performing matching. Furthermore,

property characteristics can create inferences and require the matchmaker to deduce statements that are not explicitly stated in the ontology or the service description. UDDI, however, only supports basic syntax matching. It cannot take into account semantic information presented by the ontology.

Hence there are two main challenges in providing OWL-S support using UDDI registries.
- OWL-S descriptions must be correctly expressed in the UDDI data model without losing any details.
- Ontology awareness and semantic matching must be incorporated into the querying process.

3.1. Semantic Annotation and Matchmaking

This section describes the kinds of semantic annotations and matchmaking supported by our system. For consistency, we will use the following three example ontologies in Figure 1 and the example service description and ontology concept in Figure 2 for the rest of the paper. In the ontology diagrams, ovals represent classes and squares represent properties types. Solid lines extending downwards represent instances of classes and dotted lines extending upwards represent properties that can be attached to the class. The arrows between classes and between property types represent hierarchical relationships. In the service description diagram, the property annotations that further refine the class are represented by dotted lines. The OWL document for the examples can be found in the appendix. Notice that the contactPerson property is defined as a transitive property. It will be used as the example for inferred properties through out the rest of the paper.
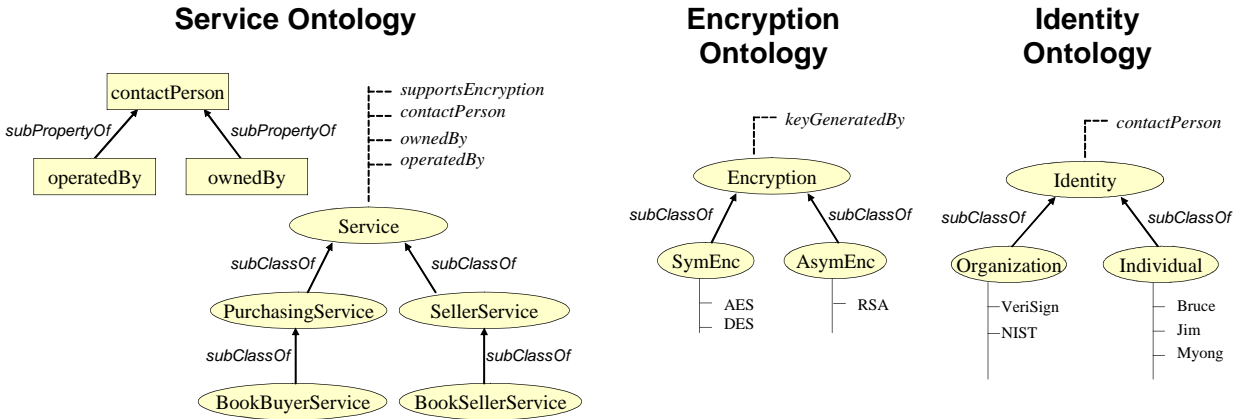
Figure 1. Diagrams for Service Ontology, Encryption Ontology, and Identity Ontology that will be used as examples throughout this paper

Semantic annotations of Web services use the concepts from ontologies. For example, a Web service may advertise itself as a BookSellerService from the Service Ontology. The service description can further refine the ontology concept by defining property annotate such as in Figure 2. Semantic annotations can have multiple layers of annotations. However, the UDDI data model is only capable of storing one layer of annotations because it was designed to deal

with flat identification and categorization systems. Thus the first challenge is to correctly express complex semantic service descriptions such as the one in Figure 2 using the UDDI data model.
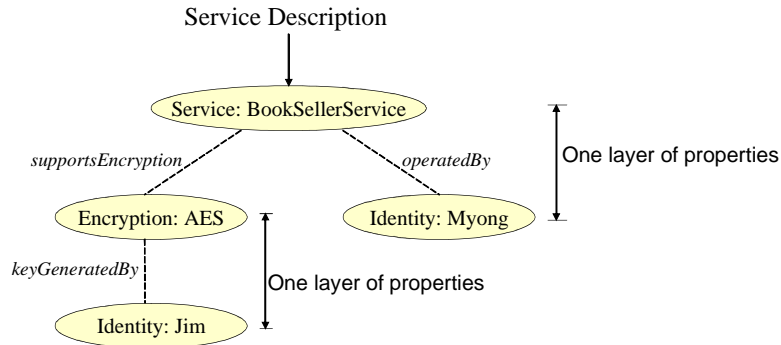
Figure 2. Diagram of a service description that will be used as an example throughout this paper

The second challenge is to provide support for semantic query. Three types of queries must be supported by the system. Detailed description of query support will be presented in section 4.

Exact Match Queries

The first type of queries uses the same concepts as those specified in the service description. The concepts in the service description and the query are syntactically identical and matching can be done syntactically. This type of query processing does not require ontology awareness.
- Find a BookSeller service
- Find a BookSeller service operated by Myong
- Find a BookSeller service that supports AES encryption
- Find a BookSeller service that supports AES encryption with key generated by Jim

Semantic Match Queries

The second type of queries uses semantically related concepts to those specified in the service description. This type of query processing requires ontology awareness. The classes, instances, and properties specified in these queries are not syntactically the same as those in the service description. However, the following queries should all match the service description in Figure 2 because they are related due to the class and property hierarchies established by the ontologies. The semantic relationships between classes, instances, and properties must be captured and taken into account by the matchmaker in order to support semantic queries.
- Find a BookSeller service that supports Encryption
- Find a SellerService that supports SymEnc with the key generated by Jim
- Find a SellerService service with contact person of Myong

Inferred Property Queries

In addition to matching classes and property types, semantic query processing also take into account inferred properties. Certain property annotations are not based on any statement explicitly expressed in the ontology. Instead, they are inferred based on reasoning about the

ontology. For example, contactPerson is defined in the ontology as a transitive property. If the instance Myong is defined to have the contactPerson property of Jim, then queries for a BookSeller services with contactPerson of Jim should also return the service description in Figure 2 because it is an inferred property. The service description does not explicitly annotate the property BookSeller with contactPerson of Jim. This property is inferred based on other definitions and annotations in the ontology and service description. The registry must resolve and capture inferred properties in order to properly perform semantic query processing.

3.2. UDDI Overview

UDDI is intended to serve as a repository for Web service descriptions. To facilitate description and discovery of Web services, the UDDI specification defines a set of data models for storing Web service descriptions and an API for interacting with the registry [6]. The core data model consists of objects describing the Web service (businessService), the service provider (businessEntity), and the service binding (bindingTemplate). In addition to the static text-based data fields, these data objects can incorporate metadata into the description by making references to tModels. TModels, or technical models, provide extensibility to the overall UDDI framework because they can be used to represent any concept or construct [6]. Examples include standards, protocols, identification systems, categorization systems, and namespaces. The versatility of the tModel comes from the fact that it only needs to serve as a place holder. Information is not actually imported into the registry. The tModel simply holds a pointer to external resources providing that information. The UDDI standard envisions two primary functions for tModel references in the context of service description.
- Represent standard interfaces, specifications, or protocols such as HTTP, SSH, and WSDL documents. References to these tModels indicate conformance to the standard.
- Represent identification or categorization systems such as NAICS, UNSPSC and U.S. Tax ID. References to these tModels indicate adherence to the systems along with specification of a specific value within the system.

Core data model objects reference tModels with keyedReference and keyedReferenceGroup objects stored under the categoryBag or identifierBag. The keyedReference object has three data fields.
- tModelKey – the unique identifier of the tModel being referenced
- keyValue – keyword within the categorization or identification system. Only necessary when referencing tModels for identification or categorization systems
- keyName – descriptive name intended for human consumption. It is optional and not used by the registry for matching in most instances

KeyedReferences to tModels imply a relationship between the data model object making the reference and the concept represented by the tModel. The specific type of the relationship depends on the concept and is not explicitly stated. Usually keyedReferences imply conformance, either to standards, or to categorization and identification systems. The keyedReferenceGroup object can be used to group logically related keyedReferences. It has a

tModelKey field and one or more instances of keyedReferences. For example, when specifying a longitude and latitude pair, the tModelKey of the keyedReferenceGroup would point to the tModel for the WSG 84 system and there would be two keyedReferences holding values for longitude and latitude respectively.

The tModel framework is very powerful and provides the UDDI system with a great deal of extensibility and flexibility. They can be used for a variety of different purposes because they can represent any concept. They will be leveraged in our system to storage of ontology information and OWL service descriptions.

3.3. Strategy for Mapping Service Description

This subsection provides a broad overview and the rationale for the mapping strategy of OWL service descriptions into the UDDI data model. Detailed description of the mapping specifications will be presented in section 5.

Each individual concept within the ontology including classes and object properties will be incorporated into the UDDI registry as a tModel object that can be referenced individually. This use of tModels is more complex than what is envisioned by the UDDI specification. In its traditional use for categorization and identification systems, the tModel serves as a place holder for the overall system. The actual structure and organization of the system is not imported into the registry. Specific concepts inside the system are referenced using keywords stored in the keyValue field of keyedReferences. For example in Figure 3, in order to make a reference to "cat" in the UNSPSC system, the categoryBag of the BusinessService will hold a keyedReference to the UNSPSC tModel with KeyValue of 10101501 [8]. The meaning of 10101501 is not imported into the registry and has to be looked up at an external resource. The UDDI search engine performs syntax matching and the meaning of the keywords is not relevant.
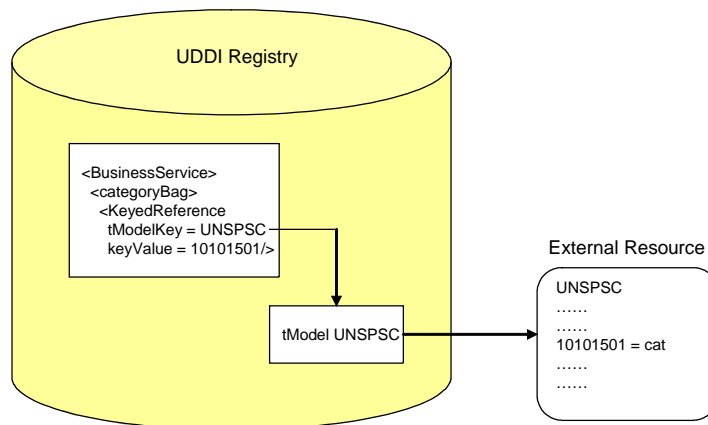
Figure 3. tModel references using traditional categorization systems.

Using tModels under the traditional UDDI scheme, the "cat" concept cannot be refined with another layer of annotation. There is no way to specify a more specific cat by stating that its

age is three months or its origin is Persia. This is because the cat concept exists as an external resource. Service descriptions are limited to the level of specificity of the categorization or identification system.

OWL concepts, on the other hand, can be annotated with additional property values as long as the specific ontology allows for it. Since the property values can be ontology classes and instances themselves, composite concepts can have multiple layers of annotations like the example in Figure 2. This is the reason that each individual concept in the ontology needs to be incorporated into the UDDI registry as tModel objects that can be referenced individually.

Anonymous instances are composite concepts defined as part of a service description. They are unnamed and do not exist in the original ontology. Figure 4 is the OWL representation of the concept in Figure 2 defined as part of a service description.


```
<profile:Profile rdf:ID="ServiceProfile1">

    <profile:serviceCategory rdf:resource="&service:BookSellerService">

    <service:supportsEncryption rdf:resource="&encryption:AES">
        <encryption:KeyGeneratedBy rdf:resource="&identity;Jim"/>
    </service:supportsEncryption>

    <service:operatedBy rdf:resource="&identity;Myong"/>
    </service:BookSeller>

</profile:Profile>
```

Anonymous Instance 1

Anonymous Instance 2
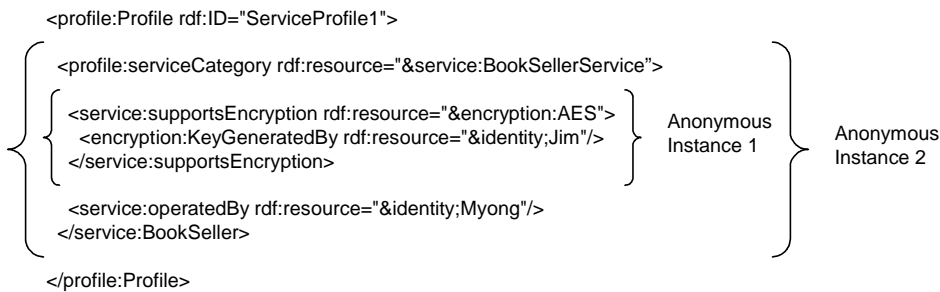
Figure 4. Example of anonymous instances defined in service descriptions

Anonymous instances cannot be referenced using keywords in the same manner as concepts in categorization or identification systems. Since they do not exist in the original ontology, they will not have agreed upon names that can be used as keywords. However, anonymous instances are distinct and need to be referenced individually. Our solution is to import the ontology into the registry and represent each class or instance with its own tModel. The base ontology concept tModels will be named and can be queried for by name. Anonymous instances defined by service descriptions will be represented by a new unnamed tModel created as extensions to the ontology. The anonymous instance tModel will reference its base class tModel using a keyedReference. The annotations are captured as keyedReferenceGroups that will hold references to the property type and property value concept tModels as child keyedReference elements. As seen in Figure 5, each tModel can hold one layer of annotations, and chaining multiple tModels together will allow for representation of composite classes with multiple layers of annotation.

```
<tModel tModelKey = "uuid:AnonymousInstance2>
  <categoryBag>
    <keyedReference tModelKey="uuid:BookSellerService"/>        ]─ Base class

    <keyedReferenceGroup>
      <keyedReference tModelKey="uuid:supportsEncryption"/>
      <keyedReference tModelKey="uuid:AnonymousInstance1"/>      }─ Property annotation
    </keyedReferenceGroup>

    <keyedReferenceGroup>
      <keyedReference tModelKey="uuid:operatedBy">
      <keyedReference tModelKey="uuid:Myong">                    }─ Property annotation
    </keyedReferenceGroup>
  </categoryBag>
</tModel>

<tModel tModelKey = "uuid:AnonymousInstance1>
  <categoryBag>
    <keyedReference tModelKey="uuid:AES"/>
    <keyedReferenceGroup>
      <keyedReference tModelKey="keyGeneratedBy"/>
      <keyedReference tModelKey="uuid:Jim"/>
    </keyedReferenceGroup>
  </categoryBag>
</tModel>
```

Figure 5. Example of anonymous instance tModels. Chaining captures multiple layers of annotation.

The entire anonymous instance can be referred to by referencing the top layer concept tModel. The following diagram in Figure 6 shows the overall structure of the UDDI data model representation the service description presented in Figure 2. New tModels, AnonymousInstance1 and AnonymousInstance2, are created to represent the composite anonymous instances. The businessService object references the overall concept by simply referencing the top level concept tModel.
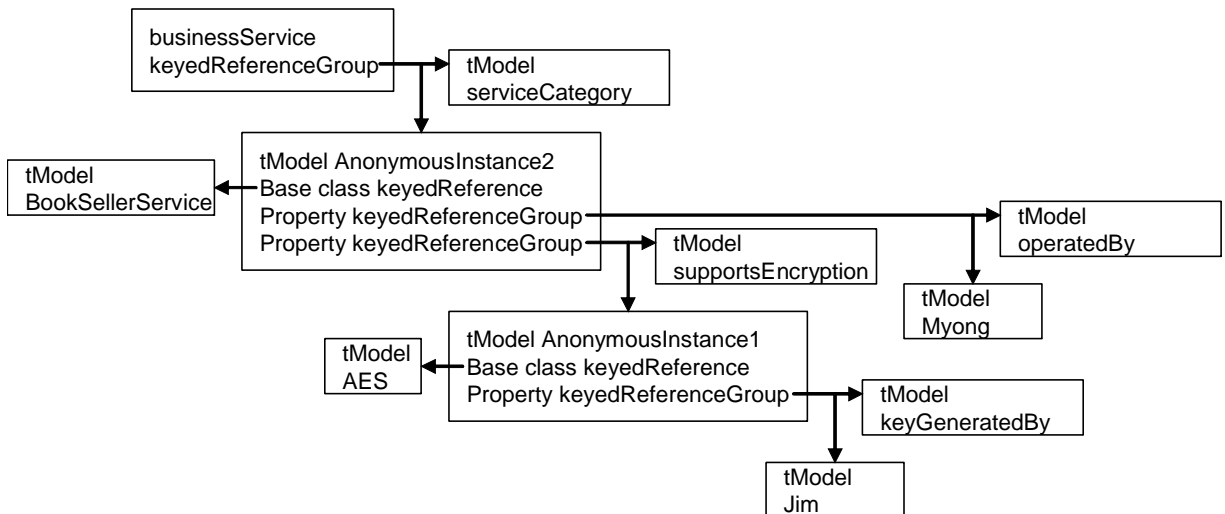
Figure 6. Mapping of the service description in Figure 2 into UDDI data models.

3.4. Strategy for Performing Semantic Matching

Since the search engine specified by UDDI is only capable of performing syntax matching, it cannot be customized to take into account the relationships between concepts in the ontologies. Ontologies establish hieratical relationships between classes, instances, and properties. For example, AES is defined as a subclass of Encryption and that relationship need to be taken into account during matchmaking. However, there is no way to tell the UDDI search engine to treat AES and Encryption as related classes so that queries for one will return the other. This information can be easily captured in the tModels for the ontology concepts. The tModel for AES can simply hold a remark with information stating that it should be returned by queries for Encryption. However, the UDDI search engine will not understand and respect that remark. There are two possible approaches for making the syntactic UDDI search engine return semantic results. Sophistication can be added during either publishing or querying.

The first approach is to fully resolve semantic ontology relationships at publishing time and index them so that they can be processed syntactically. Whenever a service description refers to a concept, it will make additional references to all its *identity concepts*. Identity concept is defined as a related concept that queries should also return when they return the original concept. If queries for the class Encryption should return the class RSA, then Encryption would be an identity class of RSA and all references to RSA will also need to reference Encryption. The identity concept approach applies to classes and instances as well as object property types. The list of identity concepts depends on the concept hierarchy established in the ontology using the OWL constructs *subClassOf*, and *equivalenceClass* for classes and instances, and *subPropertyOf*, and *equivalenceProperty* for property types. In addition, classes can be defined using complex class expressions with Boolean combinations involving the constructs *unionOf*, *intersectionOf*, and *complementOf*. An ontology reasoner must be applied at publishing time to resolve identity concepts for all classes, instances and property types.

The problem for this approach is that for anonymous instances, identity concepts can also be derived by varying the concepts and removing property definitions. For example, the following concepts are all identity concepts of the concepts described in Figure 2. That is, queries for these concepts should return the concept described in Figure 2.
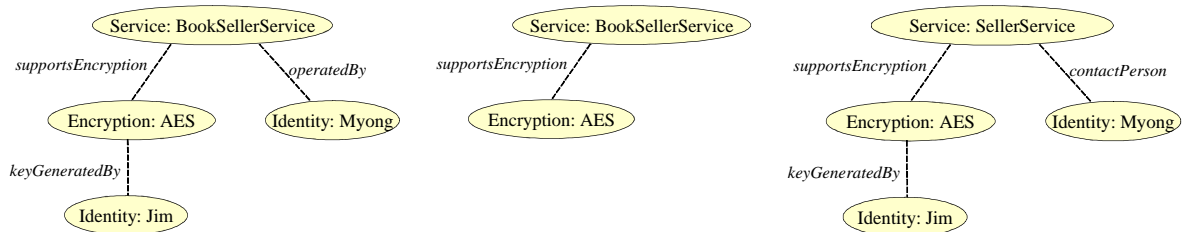
Figure 7. Diagram of example concepts that should match the concept from Figure 2.

For anonymous instances, the list of identity concepts will grow exponentially with the complexity of the concept. The list of identity concepts for anonymous instances must include not only the classes and instances already in the ontology, but also any other anonymous instances that could be potentially created by service descriptions in the future. Otherwise, whenever a new anonymous instance tModel is created, all previously created concept

10

tModels will have to be check to see if they need to reference the new tModel as an identity concept. The complexity involved in doing so makes it prohibitive. The only way to avoid modification of previously published tModels is to enumerate all possible identity concepts at the initial publication of the concepts. However, this is also impossible because the number of possible identity concepts for composite anonymous instance with multiple layers of description grows exponentially.

The other possible approach is to create sophisticated queries with the list of identity concepts that exist in the registry. This approach will be scalable because instead of having to deal with all possible identity concepts, it will only have to include ones that have already been created. The problem is that this approach is that it requires the UDDI registry to support Boolean queries. Suppose the query is for "services that support AES and are operated by Jim". If the identity concepts of AES include SymEnc and Encryption, and the identity concepts of Jim include Individual and Identity, then the BusinessServices query for UDDI is represented by the following Boolean expression.

$$(\text{supportsEncryption AND (AES OR SymEnc OR Encryption)})$$
$$\text{AND}$$
$$(\text{operatedBy AND (Jim OR Individual OR Identity)})$$

The above Boolean expression cannot be expressed as a single query in the current version of UDDI, which only allows for Boolean qualifiers over the entire query. It would be possible to simulate the Boolean expression by making multiple queries and combining the results. However, the number of queries needed grows exponentially with the number of object property annotations and the solution will not be scalable. For example, the Boolean expression above will require 9 separate queries. If the requester adds another requirement with three identity concepts, the number of queries required will increase to 27.

As you can see, it is basically impossible to perform semantic query processing using the existing UDDI search engine alone. Our approach requires the use of an additional matchmaker component on the client-side as shown in Figure 8.
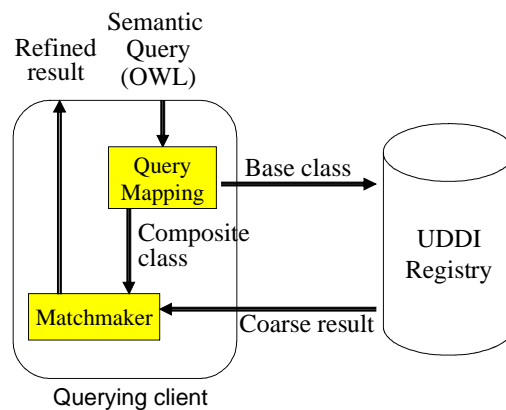
Figure 8. Diagram of the querying process

The query processed by the UDDI search engine will deal only with the base class of composite anonymous concepts and return a coarse list of possible matching service descriptions. The client-side will then refine the list by matching composite anonymous instances in their entirety. For example, if the query is for a "BookSeller that supports AES and is operated by Jim," the query passed on to UDDI will simply be for a BookSeller as shown in Figure 9. The UDDI registry will return all the service descriptions that are booksellers. The matchmaker on the client-side will then refine the results by matching them against the full composite concept of "BookSeller that supports AES and is operated by Jim."
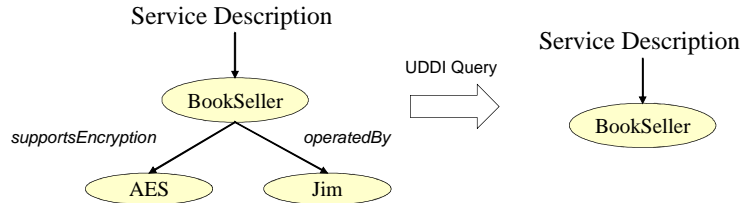
Figure 9. Query processed by UDDI

The UDDI search engine will be made to return semantic results for its part of the query processing by resolving and indexing identity concepts at publishing time. Since queries to UDDI will only use base ontology classes, the list of identity concepts can be limited to classes and instances in the original ontology and will not include anonymous instances. This approach avoids the exponential growth problem in the number of identity concepts mentioned earlier. Identity concepts for classes, instances and property types are resolved at the ontology publishing time. The relationship between the identity concept and the original concept is captured in the keyValue field of keyedReferences. This information can be used to determine the degree of match in the matchmaking process. The specifics of semantic matchmaking will be discussed later in section 6.

```
<tModel tModelKey="uuid:operatedBy">
  <name>operatedBy</name>
  <categoryBag>
    <KeyedReference keyValue="exactProperty" tModelKey=" uuid:operatedBy"/>          ─┤─ Property

    <KeyedReference keyValue="generalizationProperty" tModelKey=" uuid:contactPerson"/> ─┤─ Identity concepts of property
  </categoryBag>
</tModel>


<tModel tModelKey="uuid:SellerService ">
  <name>SellerService</name>
  <categoryBag>
    <KeyedReference keyValue="exactRelationship" tModelKey="uuid:SellerService"/>      ─┤─ Class

    <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Service"/>
    <KeyedReference keyValue="specializationRelationship" tModelKey="uuid:BookSeller"/> ─┤─ Identity concepts of class
  </categoryBag>
</tModel>
```

Figure 10. Identity concepts stored by ontology concept tModels

Whenever service descriptions or concept tModels reference an ontology concept, references to its identity concepts must also be made. This can be done syntactically by simply copying over the identity concepts in the concept tModel being referenced. This way, the service

description will be returned by the UDDI search engine for queries involving any of the identity concepts. This applies to all ontology references including base class references as well as annotation references. For base class references, identity concepts are captured as keyedReferences parallel to the exact concept reference such as in Figure 10. Annotation references will be captured inside keyedReferenceGroups such as in Figure 11. The matching rules for keyedReferenceGroup require the child keyedReferences specified in the query to be a subset of the child keyedReferences in the service descriptions. This set operation is effectively the AND Boolean operator. Property type and property value tModels can be stored in parallel keyedReferences inside keyedReferences without any special designation. The query can simply reference a property type tModel and a property value tModel inside the keyedReferenceGroup and they will correctly match annotations in service descriptions according to UDDI matching rules.

```
<BusinessService>
  <name>ServiceProfile1</name>
  <categoryBag>
    <KeyedReferenceGroup>
      <KeyedReference keyValue="exactProperty" tModelKey="uuid:profile:serviceCategory">      ]- Property type

      <KeyedReference keyValue="exactRelationship" tModelKey="uuid:SellerService"/>           ]- Property value concept

      <KeyedReference keyValue="specializationRelationship" tModelKey="uuid:BookSellerService"/>  ]
      <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Service"/>              }  Identity concepts
    </KeyedReferenceGroup>                                                                          ]  of property value
</categoryBag>
</tModel>
```

```
<tModel tModelKey = "uuid:AnonymousInstance2>
  <categoryBag>
    <keyedReference keyValue="exactRelationship" tModelKey="uuid:AES"/>                       ]- Base class

    <keyedReference keyValue="generalizationRelationship" tModelKey="uuid:SymEnc"/>           ]  Identity concepts of
    <keyedReference keyValue="generalizationRelationship" tModelKey="uuid:Encryption"/>       ]  base class

    <keyedReferenceGroup>
      <keyedReference keyValue="exactProperty" tModelKey="operatedBy"/>                       ]- Property type

      <keyedReference keyValue="generalizationProperty" tModelKey="contactPerson"/>           ]- Identity concept of property type

      <keyedReference keyValue="exactRelationship" tModelKey="uuid:Myong"/>                   ]- Property value

      <keyedReference keyValue="generalizationRelationship" tModelKey="uuid:Individual"/>     ]  Identity concepts of property value
      <keyedReference keyValue="generalizationRelationship" tModelKey="uuid:Identity"/>       ]
    </keyedReferenceGroup>
  <categoryBag>
</tModel>
```

Figure 11. Identity concepts are copied whenever ontology concepts are referenced

The client-side matchmaker also makes use of the identity concept information for semantic matching. The matchmaker will compare the concept specified in the query with all the identity concepts of the service description concept at each level of annotation. The result will be the same as a semantic match between the two concepts. However, the matchmaker will operate syntactically and it will not need to be ontology aware. The client-side matchmaker will be discussed in detail in section 6.

3.5. Strategy for Resolving Inferred Properties

In addition to the class and property hierarchies established in the ontology, inferences based on property characteristics must also be taken into account in performing semantic matchmaking. These constructs in the OWL language are used for higher level reasoning about the ontology. Certain facts that are not explicitly stated can be inferred based on other definitions in the ontology. Our system will fully support inferences based on transitive and symmetric property characteristics. Other forms of inference will not be supported due to the limitation of the UDDI framework and their lack of direct relevance to service descriptions and matchmaking. Details of OWL support in our system will be discussed later in section 3.7.

Object properties can be defined with the property characteristics of *TransitiveProperty* and *SymmetricProperty*. Inferred properties based on these two property characteristics will be resolved and added as property annotations for all the base ontology concepts at the publishing time of the ontology by ontology reasoner. For symmetric properties, the inverse of the property annotation definition must be added. For transitive properties, the entire chain of transitivity must be resolved and added to the property annotation. Furthermore, the chain of transitivity must be extended when new anonymous instance tModels or service descriptions are annotated using transitive properties. As a result, special operations must be performed specifically for property annotations using transitive properties during publication of service description. Those operations will be discussed in detail in section 5. No differentiation will be made between explicit and inferred properties for matchmaking. From the ontology perspective, inferred properties are just as valid as explicit property annotations. Service descriptions in OWL reconstructed from UDDI data structures will explicitly state all inferred property annotations. However, the reconstructed document will be semantically identical to the original OWL service description.

3.6. System Architecture

Figure 12 shows the overall system architecture of our prototype implementation. The shaded boxes are the four add-on modules that will reside on the client-side. Only clients wishing to use UDDI as a semantic registry will need to add these modules to their machines. Other clients use the UDDI registry normally.

Query

Semantic
Query
(OWL)

Query
Mapping

Client-side
Matchmaker

Ontology publisher

Ontology
(OWL)

Ontology
Mapping

UDDI
API

UDDI
Registry

UDDI
API

Service
Description
(OWL)

Service
Description
Mapping

Service description publisher
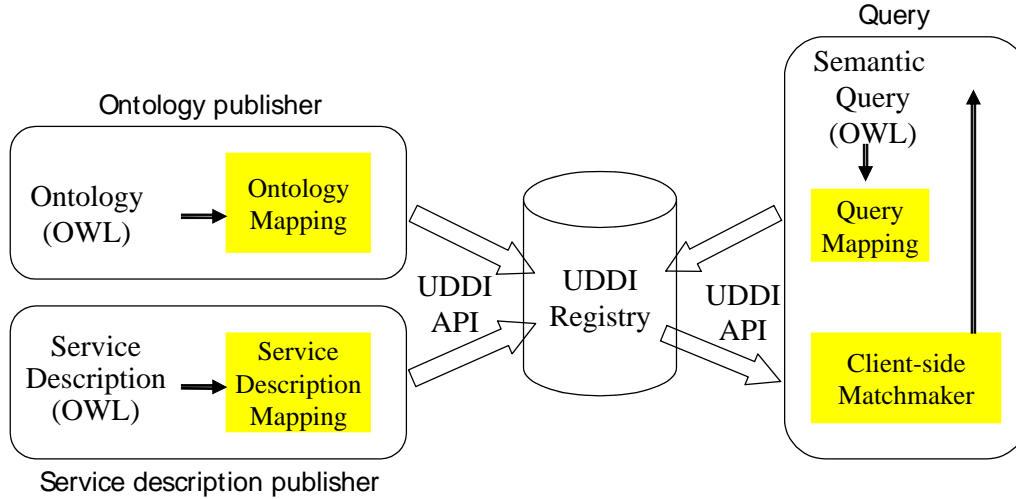
Figure 12. System architecture of our prototype.

Ontology mapping and service description mapping modules are implemented in XSLT and Java. XSLT translates OWL documents into the UDDI data model and Java code publishes them into the registry using the UDDI client-side API. Identity concepts and inferred properties are resolved in the ontology mapping module which includes a simple ontology reasoner implemented in XSLT. The service description mapping module will syntactically propagate semantic information captured in the ontology tModels to the service descriptions.

The query mapping module is also implemented in XSLT and Java. The XSLT component will strip annotations from composite concepts and translate OWL queries into UDDI queries. The Java component will then query the registry through the UDDI client-side API. The results returned by the registry will be the businessServices objects with references that match only the base ontology concepts.

The client-side matchmaker is a Java module that will refine the query results by performing matchmaking that takes into account all the annotations of composite concepts. It will query the registry for concept tModels referenced by the businessService object to fully reconstruct the service description. Then it will match the service description with the query by examining the concepts at each layer of annotation. This component is only necessary because the UDDI specification lacks support for Boolean queries. Its task can be folded into the registry if future versions of UDDI provide that support. The need for the client side matchmaker will be discussed further in section 7.

3.7. OWL Language Support

Our prototype system will not support all the constructs of the OWL language [9]. This is governed by the functional limitations of UDDI and the desire to keep the solution simple. However, the most important aspects of OWL are supported including all the functionalities that are commonly used for service descriptions and matchmaking.

The identity concept approach captures the overall class and property hierarchies expressed in the ontology. The mapping from OWL into the UDDI data structure supports service descriptions and classes that are annotated with properties in multiple layers. This includes anonymous instance annotations defined by service descriptions as well as class annotations using restriction classes defined by ontologies. Inferences based on transitive and symmetric property characteristics are supported. The following OWL language constructs will be fully supported:

- *Class* definition
- *ObjectProperty* definition
- *DatatypeProperty* definition
- Annotation of concepts using properties
- *subClassOf* and *equivalentClass* relationship
- *subPropertyOf* and *equivalentProperty* relationship
- *Restriction* classes with *onProperty* and *hasValue* attributes
- *TransitiveProperty* and *SymmetricProperty* attributes
- *UnionOf* class expressions

The following components of the OWL language are not supported by our prototype implementation of the system.

- The system will not enforce restrictions laid out in the ontology. OWL constructs relating to restrictions such as *domain*, *range*, *minCardinality*, *maxCardinality*, *cardinality*, *allValuesFrom*, and *someValuesFrom* will not be processed.

- Queries involving cardinality of properties will not be supported. We expect that cardinality queries will be relatively uncommon for matchmaking of Web services. Furthermore, the UDDI specification simply does not include the concept of cardinality and providing this support would be very hard.

- The system will not support property characteristics used primarily for ontology reasoning such as the OWL constructs *inverseOf*, *FunctionalProperty*, and *InverseFunctionalProperty*. These property characteristics are rarely used by descriptive ontologies that deal with Web services and do not contribute to matchmaking. A design decision was made to limit the complexity of the system.

- Boolean combination of class expressions other than *unionOf* will not be supported. This is due to the limitation of UDDI in terms of support for Boolean queries and a design decision to limit complexity of the system. OWL constructs *disjointWith*, *equivalentClass*, *complementOf*, and *intersectionOf* will be ignored. There is simply no way to properly express these Boolean combinations using the UDDI data model. The default treatment of multiple entries by the UDDI search engine is with the AND operator. This is equivalent to the union set operation. Therefore, the only Boolean combination that is supported is the *unionOf* construct.

- Classes and instances published as part of service descriptions will not affect the original ontology from the point of view of other service descriptions. In effect, service descriptions can be completely insulated from each other. Relationships they establish and modifications they make to the underlying ontology using anonymous instances will not have any influence on other service descriptions as shown in Figure 13. Anonymous instances are not treated as true extensions of the ontology, but rather as additional concepts for use by the particular service description. Other service description can choose to utilize the additional concepts. However, the underlying ontology will always remain intact. For example, if a service description declares a new class, PurchaserService, as an equivalent class of BuyerService, that equivalency is only applied to the new class. PurchasingService will hold identity references to BuyerService. However, the base ontology will not be changed and BuyerService will not be modified to include an identity concept reference to PurchaserService. Queries for BuyerService will correctly return PurchasingService. Queries for PurchaserService, on the other hand, will not return BuyerService. PurchaserService is not part of the base ontology and its queries will be processed correctly for service descriptions that explicitly use it. There are two main reasons for this design decision. The first reason is the supposition that the publisher of the ontology should be the only one with the privilege to make modifications to the base ontology. If this were not the case, the base ontology could be maliciously or accidentally corrupted by publishers of service descriptions that are presumed to be less trustworthy. For example, a service description can declare that BuyerService and SellerService are equivalent classes. This would make the ontology inconsistent and queries based on the modified ontology will return incorrect results. The second reason deals with difficulty of providing for this support. The complexity in having to check the consistency of the ontology and modify previously published concept whenever anonymous concepts are published makes its implementation very difficult and impractical.
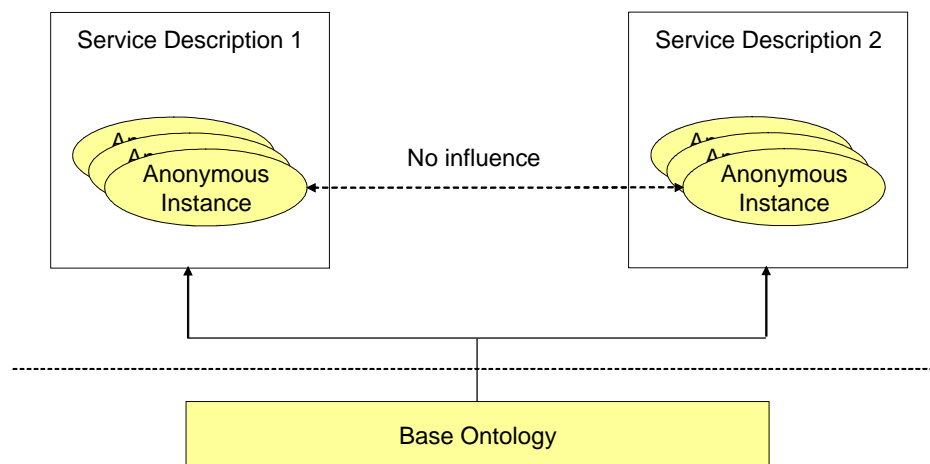

Figure 13. Extensions to the ontology from anonymous instances do not change the base ontology and will not influence other anonymous instances.

4. Query Support

Several different types of queries will be supported by the system. The requirements for query support will govern the specifics of the mapping from OWL to the UDDI data model. Extra data and labels will be stored to provide support for specific categories of queries. Four categories of queries will be supported by the system. The first two categories deal with finding tModels of ontology concepts. They can be used to browse the ontologies and reconstruct them from the data stored in the UDDI registry. The last two types of queries deal with finding BusinessService objects representing service descriptions. They are used to actually locate Web services.

Ontology Query

This category of queries allows users to retrieve all tModels associated with a specific ontology from the UDDI registry. In order to allow for this query, all ontology concepts will reference a single overview tModel that is created to represent the overall ontology. A query for tModels with references to that ontology tModel will return all the concepts associated with the ontology. The user needs to be able to either retrieve only the base ontology tModels, or all the ontology tModels including anonymous instance defined by the service descriptions. Furthermore, all ontology overview tModels will have a special designation. This way, users can query for a list of all the ontology overview tModels in the registry.

Ontology Concept Query

This category of queries allows users to find the tModel for a specific concept in the ontology such as property types, classes, and instances. The concepts can be either a base ontology concept defined by the ontology, or anonymous instance defined by service descriptions. The base ontology concept tModels will carry the same name as the ontology class, instance or property type it represents. Consequently, they can be queried for directly by name. Anonymous instance tModels, on the other hand, are unnamed and must be queried for using their base classes. Using this method, the UDDI search engine will return all the tModels with a particular base class and the client-side matchmaker will refine the results by resolving and matching the concepts in their entirety. To support semantic matching, it is necessary for all class and instance tModels to hold a list of base ontology identity classes. The relationship between the identity classes and the original concept must also be captured so the query can be customized. For example, if the query is for AES, the user should be able to require an exact match, in which case only AES would be returned, or only require a partial match, in which case Encryption would also be returned. Hence it is possible for multiple tModels to match a single query. The specific matchmaking rules will be discussed later in section 6.

Complete Service Description Query

This category of queries allows users to query for service descriptions by specifying annotations in their entirety. Ontology references in the service description are attached to the top level service with all the property types and property values fully defined such as in Figure 14. The users are basically querying with a hypothetical description of their desired service. The query forwarded to the UDDI will only contain the base classes of anonymous instances. Hence the BusinessService object will need to also reference the base class of any

anonymous instances it uses as property values. In order to have semantic matching, all the identity concepts of the base class must be included as references in the BusinessService object as well. Again, the relationship between the identity classes and the original concept will be captured so the query can be customized. The matchmaker on the client-side will refine the results by resolving and matching the composite concepts in their entirety.
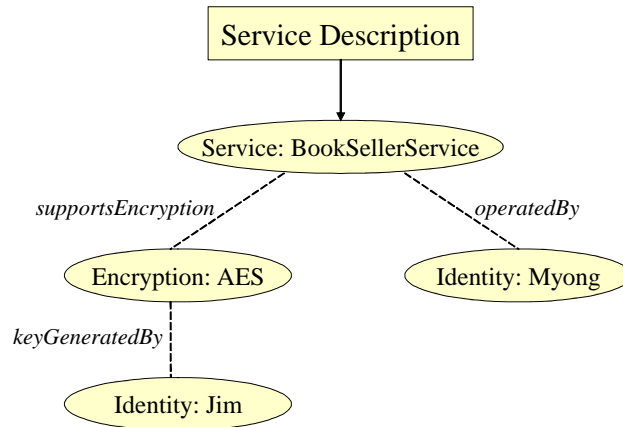
Figure 14. Structure of complete service description queries

Partial Service Description Query

This category of queries allows users to query using snippets of annotation instead of fully defined service descriptions such as in Figure 15. They are for users that want to find services that reference particular concepts but do not care how those concepts are connected to the service descriptions. For example, the users could look for any services operated by Myong regardless of the other details of the service.
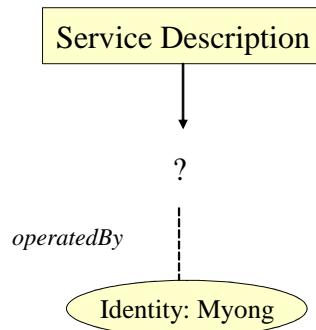
Figure 15. Structure of partial service description queries

Special data structures will be created in UDDI to support this type of queries. A special *tModelsUsed* keyedReferenceGroup in the BusinessService object will hold references to all the concepts used as property values as well as property types. The query can involve either a combination of the property value concept and the property type, or simply a property value concept. Both pieces of information will be stored and the user has the option to use them both, or ignore the property type by using a wildcard in the query.

5. Mapping Specifications for Publication

This section describes the specific mapping from OWL ontology and service description to the UDDI data model. All tModels in this section will reference UDDI type tModels so they can be used as if they were categorization systems. For the sake of simplicity, concept names in the OWL descriptions will take the form of "ontology:concept" in the following examples. Similarly, the tModelKeys for ontology concept tModels will also be denoted in the form of "ontology:concept". Actual tModelKeys used by the UDDI registry will be arbitrary unique strings that have to be determined through querying the registry.

5.1. Bookkeeping tModels

Several tModels are necessary for bookkeeping purposes as part of the mapping from OWL to the UDDI data model. They will be used as special designations by the various processors in client-side module.

Base Ontology Designation tModels

This tModel is used to designate the status of ontology related tModels in the registry. The tModel is named "isOntologyCore" and Figure 16 shows its content. In the context of ontology concept tModels, it is used to differentiate between concepts that are defined by the ontology and anonymous instances defined by service descriptions. This tModel will be used as a value set to indicate whether a concept tModel is created as part of the ontology. TModels defined as part of the ontology will reference this tModel with the keyValue of "true". TModels for anonymous instances defined by a service description will reference this tModel with the keyValue of "false". Queries using this tModel and the appropriate keyValue can differentiate between concepts in the original ontology and concepts created as part of service descriptions.

```
<tModel tModelKey="uuid:isOntologyCore">
  <name>isOntologyCore</name>
  <categoryBag>
    <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

Figure 16. "isOntologyBase" tModel

OWLType tModel

This tModel is used to designate whether a concept tModel represent an ontology, a class, or a property type. Ontology tModels will reference this tModel with the keyValue of "ontology". Class and instance tModels will reference this tModel with the keyValue of "class". Property type tModels will reference this tModel with the keyValue of "property".

```
<tModel tModelKey="uuid:owlType">
  <name>owlType</name>
  <categoryBag>
    <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

Figure 17. "owlType" tModel

PropertyDefinition tModel

The "propertyDefinition" tModel is referenced by the tModelKey of KeyedReferenceGroups that define property annotations. Figure 18 shows the content of this tModel.

```
<tModel tModelKey="uuid:propertyDefinition">
  <name>propertyDefinition</name>
  <categoryBag>
    <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyValue="categorizationGroup" tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

Figure 18. "propertyDefinition" tModel.

Property Characteristic tModel

This tModel will be used to designate property characteristics of tModels. TModels for property types with property characteristics will reference this tModel and use the property characteristic as the keyValue. Special operations need to be performed during ontology publication for property types that are transitive or symmetric. References to this tModel in the property type tModel will indicate that those operations need to be performed.

```
<tModel tModelKey="uuid:propertyCharacteristic">
  <name>propertyCharacteristic</name>
  <categoryBag>
    <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

Figure 19. "propertyCharacteristic" tModel.

TModelsUsed tModel

In order to support the forth categories of queries, partial service description queries, BusinessService objects will need to keep track of all the tModels it references either directly or indirectly. For example, if the BusinessService references the anonymous instance "BookSeller operated by Jim", it will need to make references to the tModel for both BookSeller and Jim. The list of tModels will be kept by BusinessService objects in a special keyedReferenceGroup. This "tModelUsed" tModel will be referenced as the tModelKey of that keyedReferenceGroup. Figure 20 shows the content of the "tModelUsed" tModel.

```
<tModel tModelKey="uuid:tModelsUsed">
  <name>tModelsUsed</name>
  <categoryBag>
   <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
   <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>
   <keyedReference keyValue="categorizationGroup" tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

Figure 20. "tModelsUsed" tModel.

5.2. Translation of the Ontology

There will be four types of tModels associated with the translation of OWL ontologies into the UDDI data model. Each of these tModels will make keyedReferences to UDDI tModels for categorization systems so their references can hold a KeyValue. In addition to user defined ontologies, the OWL-S ontologies must also be imported into the registry.

Ontology tModel

The ontology overview tModel will serve as a place holder and namespace for the ontology as a whole. It will hold overview information including the ontology name, description, and URL of external descriptions analogous to tModels for categorization or identification systems. This tModel will be referenced by all instance, class and property tModels associated with the ontology using the KeyValue of "ontologyReference". Figure 21 summarizes the translation of ontology overview tModels.

```
<!ENTITY service "http://ontology.com/service.owl">
<owl:Ontology>
  <rdfs:comment>A Service Ontology</rdfs:comment>
</owl:Ontology>
```



```
<tModel tModelKey="uuid:service">
  <name>service</name>
  <description>A Service Ontology</description>
  <overviewURL>http://ontology.com/service.owl</overviewURL>
  <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
  <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>

  <keyedReference keyValue="ontologyReference" tModelKey="uuid:isOntologyCore"/>  ⊢ Ontology overview designation
</tModel>
```

Figure 21. Translation of the ontology tModel

Property Type tModels

The property type tModel will store property information defined in the ontology. The name of the tModel will be set to the name of the property defined in the ontology. It will hold a keyedReference to the ontology tModel to indicate which ontology it belongs to. It will also hold a keyedReference to the ontology core designation tModel with keyValue of "true" indicating that it is a core ontology tModel. In addition, it will hold keyedReferences to other ontology property type tModels representing its identity concepts. All properties will have an identity concept to itself with the relationship type of "exactProperty". The other possible values are "equivalentProperty," "generalizationProperty" and "specializationProperty". The

22

list of identity properties is derived from the ontology property hierarchy determined by the ontology reasoner, which will be discussed in detail later in section 6.

Properties with property characteristics of symmetricProperty or transitiveProperty will reference the respective tModels. These references will be used by the mapping modules. The domain and range restrictions for property types will not be enforced by UDDI. It will be up to the users who created the OWL-S documents to correctly assign property attributes. However, this information will be captured in the tModel for the sake of completeness. The domain and range information for the property is held in keyedReferences with the keyValue of "domain" and "range" respectively. Figure 22 summarizes the translation of property type tModels.

```
<owl:ObjectProperty rdf:ID="contactPerson">
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Service"/>
        <owl:Class rdf:about="&identity;Identity"/>
      </owl:unionOf>
    </owl:class>
  <rdfs:range rdf:resource="&identity;Identity"/>
</owl:ObjectProperty>
```



```
<tModel tModelKey="uuid:contactPerson">
  <name>contactPerson</name>
  <categoryBag>
    <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>

    <KeyedReference keyValue="ontologyReference" tModelKey="uuid:service"/>                       } Ontology reference

    <KeyedReference keyValue="true" tModelKey="uuid:isOntologyCore"/>                             } Ontology core

    <KeyedReference keyValue="exactProperty" tModelKey=" uuid:contactPerson"/>
    <KeyedReference keyValue="specializationProperty" tModelKey=" uuid:operatedBy"/>             } Identity concepts
    <KeyedReference keyValue="specializationProperty" tModelKey=" uuid:ownedBy"/>

    <KeyedReference keyValue="transitiveProperty" tModelKey="uuid:propertyCharacteristic"/>     } Property characteristic

    <KeyedReference keyValue="domain" tModelKey="uuid:Service"/>
    <KeyedReference keyValue="domain" tModelKey="uuid:Identity"/>                                } Property domain and range
    <KeyedReference keyValue="range" tModelKey="uuid:Identity"/>
  </categoryBag>
</tModel>
```

Figure 22. Translation of property type tModels.

Simple Class and Instance tModels

The simple class and instance tModels will store information of concept defined by ontologies that do not have any property annotations. The name of the tModel will be set to the name of the class or instance defined in the ontology. It will hold a keyedReference to the ontology tModel to indicate which ontology it belongs to. It will also hold a keyedReference to the ontology core designation tModel with keyValue of "true" indicating that it is a core ontology tModel. In addition, it will hold keyedReferences to other ontology class tModels representing its identity classes. The type of relationship is stored in the keyValue field of the keyedReferences. All classes and instances will have an identity relationship to itself with the

relationship type of "exactRelationship". The other possible values are "equivalentRelationship," "generalizationRelationship" and "specializationRelationship". The list of identity classes and instances is determined by the ontology reasoner based on the ontology class hierarchy and class expressions definitions. The ontology reasoner will be discussed in detail later in section 6. Figure 23 summarizes the translation of ontology class and instance tModels.

```
<owl:Class rdf:ID="BookTraderService">
  <rdfs:subClassOf rdf:resource="#Service"/>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#BookSellerService" />
    <owl:Class rdf:about="#BookBuyerService" />
  </owl:unionOf>
</owl:Class>
```

```
<tModel tModelKey="uuid:SellerService ">
  <name>BookTraderService</name>
  <categoryBag>
    <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>

    <KeyedReference keyValue="ontologyReference" tModelKey="uuid:service"/>    ⎫— Ontology reference

    <KeyedReference keyValue="true" tModelKey="uuid:isOntologyCore"/>    ⎫— Ontology core

    <KeyedReference keyValue="exactRelationship" tModelKey="uuid:BookTraderService"/>
    <KeyedReference keyValue="equivalentRelationship" tModelKey="uuid:BookSellerService"/>
    <KeyedReference keyValue="equivalentRelationship" tModelKey="uuid:BookBuyerService"/>
    <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Service"/>     Identity concepts
    <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:SellerService">
    <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:BuyerService">
  </categoryBag>
</tModel>
```

Figure 23. Translation of class and instance tModels.

Composite Concept TModels

Composite concepts are classes and instances with property definitions. They can be defined in the ontology as restriction classes or in service descriptions as anonymous instances. They will hold a keyedReference to the ontology tModel to indicate which ontology it belongs to. The identity concepts will be exactly the same as those of their base classes with the exception of a reference to itself using the keyValue of "specificRelationship". The reference to the base class will have the keyValue of "exactRelationship". Property definitions are captured as keyedReferenceGroups. The property type and property value tModels as well as the tModels of their identity concepts are captured as keyedReferences under the keyedReferenceGroup.

For restriction classes, the tModel name will be set to the name of the class defined in the ontology. It will also hold a keyedReference to the ontology core designation tModel with keyValue of "true" indicating that it is a core ontology tModel. In addition to the identity concepts, the ontology reasoner must also resolve inferred properties based on symmetric and transitive properties for restriction class tModels. Additional property annotations need to be created for inferred properties. The list of inferred properties is resolved by the ontology reasoner, which will be discussed in detail in section 6. Both explicit and inferred properties

are captured the same way and no distinction is made between them. Figure 24 summarizes the translation of ontology restriction class tModels.

```
<owl:Class rdf:ID="JimSellerService">
  <rdfs:subClassOf rdf:resource="#SellerService"/>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="operatedBy"/>
    <owl:hasValue>identity:Jim</owl:hasValue>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```
<tModel tModelKey="uuid:JimSellerService ">
  <name>JimSellerService</name>
  <categoryBag>
   <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
   <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>

   <KeyedReference keyValue="ontologyReference" tModelKey="uuid:service"/>        ⎤─ Ontology reference

   <KeyedReference keyValue="true" tModelKey="uuid:isOntologyCore"/>              ⎤─ Ontology core

   <KeyedReference keyValue="specificRelationship" tModelKey="uuid:JimSellerService"/>
   <KeyedReference keyValue="exactRelationship" tModelKey="uuid:SellerService"/>
   <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Service"/>      identity concepts of base class
   <KeyedReference keyValue="specializationRelationship" tModelKey="uuid:BookSellerService"/>

   <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
    <KeyedReference keyValue="exactProperty" tModelKey="uuid:Service:operatedBy"/>
    <KeyedReference keyValue="generalizationProperty" tModelKey="uuid:Service:contactPerson"/>   property type identity concepts

    <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Jim"/>
    <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Individual"/>      property value identity concepts
    <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Identity"/>
   </KeyedReferenceGroup>
  </categoryBag>
</tModel>
```

Figure 24. Translation of restriction class tModels.

Anonymous instances defined by service descriptions are translated the same way as restriction classes defined in the ontology with a few exceptions. Anonymous instance are not named and the name of the tModel will be left blank. It will also hold a keyedReference to the ontology core designation tModel with keyValue of "false" indicating that it is a not core ontology tModel. Composite concepts can have multiple layers of annotations. If the annotations are composite concepts themselves, new tModels need to be created for them as well. Figure 25 summarizes the translation of anonymous instance tModels.

```
<profile:Profile rdf:ID="ServiceProfile1">
  <profile:serviceCategory rdf:resource="&service:BookSellerService">
    <service:supportsEncryption rdf:resource="&encryption:AES">
      <encryption:KeyGeneratedBy rdf:resource="&identity;Jim"/>
    </service:supportsEncryption>
  <service:operatedBy rdf:resource="&identity;Myong"/>
</profile:Profile>
```

                                    Anonymous   Anonymous
                                    Instance 1   Instance 2

```
<tModel tModelKey="uuid:AnonInst2">    ]- Anonymous instance tModel names are generated by the mapping module
  <name></name>
  <categoryBag>
    <keyedReference keyValue="categorization" tModelKey="uddi:uddi.org:categorization:types"/>
    <keyedReference keyValue="valueSet" tModelKey="uddi:uddi.org:categorization:types"/>

    <KeyedReference keyValue="ontologyReference" tModelKey="uuid:service"/>

    <KeyedReference keyValue="false" tModelKey="uuid:isOntologyCore"/>              ]- Not ontology core

    <KeyedReference keyValue="exactRelationship" tModelKey="uuid:BookSellerService"/>
    <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:SellerService"/>
    <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Service"/>

    <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <KeyedReference keyValue="exactProperty" tModelKey="uuid:Service:supportsEncryption"/>

      <KeyedReference keyValue="specificRelationship" tModelKey="uuid:AnonInst1"/>
      <KeyedReference keyValue="exactRelationship" tModelKey="uuid:AES"/>
      <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:SymEnc"/>
      <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Encryption"/>
    </KeyedReferenceGroup>

    <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <KeyedReference keyValue="exactProperty" tModelKey="uuid:Service:operatedBy"/>
      <KeyedReference keyValue="generalizationProperty" tModelKey="uuid:Service:contactPerson"/>
      <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Myong"/>
      <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Individual"/>
      <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Identity"/>
    </KeyedReferenceGroup>
  </categoryBag>
</tModel>

<tModel tModelKey="uuid:AnonInst1">  ←
  …
</tModel>
```

Figure 25. Translation of anonymous instance tModel

If restriction class property annotations are made with property types that possess symmetric or transitive property characteristic, the ontology reasoner will need to resolve inferred properties for all the classes involved. The ontology reasoner will be discussed in detail in section 6. During publication of the ontology, inferred properties are published the say way as explicit property definitions. Special operations must be performed during publication of service descriptions for transitive properties. If an anonymous instance has property annotations using transitive property types, then the property value concepts must be checked to see if they hold annotations using the same property type. If they do, the property value concepts in that concept annotation must also be copied as inferred properties to propagate transitive property characteristic. Figure 26 summarizes the translation of ontology concepts with transitive property annotations. No special operations are necessary for symmetric properties during service description publication.

```xml
<owl:Class rdf:ID="MyongSellerService">
  <rdfs:subClassOf rdf:resource="#SellerService"/>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="contactPerson"/>
    <owl:hasValue>identity:Myong</owl:hasValue>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Jim">
  <rdfs:subClassOf rdf:resource="#Individual"/>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="contactPerson"/>
    <owl:hasValue>identity:Bruce</owl:hasValue>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```xml
<owl:Class rdf:ID="Myong">
  <rdfs:subClassOf rdf:resource="#Individual"/>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="contactPerson"/>
    <owl:hasValue>identity:Jim</owl:hasValue>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```xml
<tModel tModelKey="uuid:MyongSellerService ">
 <name>MyongSellerService</name>
 <categoryBag>
  …

  <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
   <KeyedReference keyValue="exactProperty" tModelKey="uuid:Service:contactperson"/>

   <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Myong"/>
   <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Individual"/>
   <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Identity"/>

   <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Jim"/>
   <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Bruce"/>
  </KeyedReferenceGroup>
 </categoryBag>
</tModel>
```

explicit property values

inferred property values

```xml
<tModel tModelKey="uuid:Myong">
 <name>Myong</name>
 <categoryBag>
  …

  <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
   <KeyedReference keyValue="exactProperty" tModelKey="uuid:Service:contactperson"/>

   <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Jim"/>
   <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Individual"/>
   <KeyedReference keyValue="generalizationRelationship" tModelKey="uuid:Identity"/>
   <KeyedReference keyValue="exactRelationship" tModelKey="uuid:Bruce"/>
  </KeyedReferenceGroup>
 </categoryBag>
</tModel>
```

transitive property value
(determined by ontology reasoner)

Figure 26. Applying transitivity for transitive property annotations

In addition to objectProperty property types, composite concepts can also be annotated using dataProperty property types. DataProperties do not reference ontology concepts, but rather literal data types. These annotations are stored the same way as objectProperties with the exception that property value keyedReference will reference the UDDI "general_keyword" tModel and the keyValue will hold the value of the literal data type as a string.

27

5.3 Translation of the Service Description

Translation of the service description involves two steps. First, tModels for any anonymous composite concept defined in the service description must be published into the registry unless they already exist. It is important to note that anonymous instance tModels can be reused between service descriptions. Before creating another tModel, the registry is queried for the anonymous instance and the existing tModel should be used.

Second, the service description must be translated into a corresponding UDDI businessEntity and businessService objects. It is important to note that publication of service descriptions do not need to use the ontology reasoner. The identity concepts and inferred properties are fully resolved during publication of the ontology. Publication of the service description into BusinessService objects and anonymous instances can be published by making use of the semantic information already captured. This is a completely syntax based process and does not need to be ontology aware.

BusinessService Object

Information in the OWL-S service description that maps to fields in the UDDI BusinessEntity and BusinessService data model objects can be translated directly.
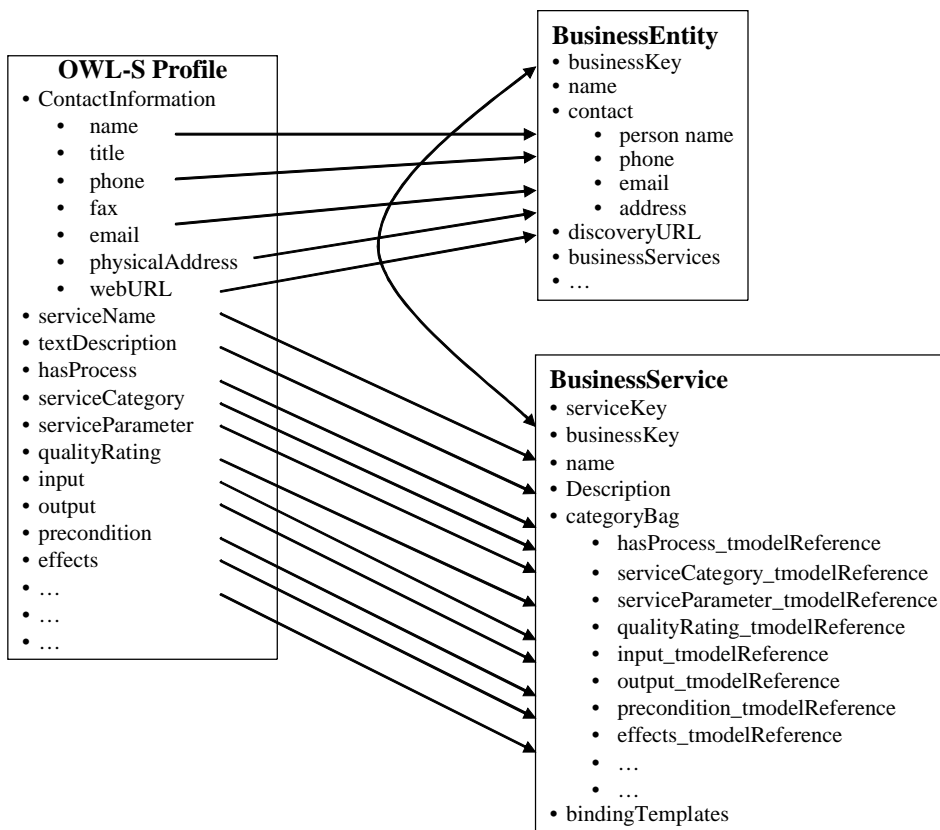

Figure 27. Diagram of the translation of service description BusinessService objects.

Ontology references in the service description are stored as keyedReferenceGroup in the same way as property annotations in the tModels for composite concepts. They are held in keyedReferenceGroups with child element keyedReferences that store the reference type and referenced concept. Ontology references that use transitive properties also need to be processed to propagate the chain of transitivity the same way it is done for anonymous instances. In addition to the ontology references, a special keyedReferenceGroup will hold the list of tModels used by the service description to facilitate the forth category of queries, partial service description queries. The tModelKey of the keyedReferenceGroup will reference the tModelsUsed tModel. The keyedReferences under this keyedReferenceGroup will list all the concepts used directly or indirectly to annotate the composite class along with the property types of the annotation. The tModelKeys of property types are stored in the keyValue fields, and the tModelKeys of the property value concept are stored in the tModelKey fields. Identity concepts for tModelsUsed will not be referenced in this prototype. Therefore, partial service description queries will have limited semantic capabilities.
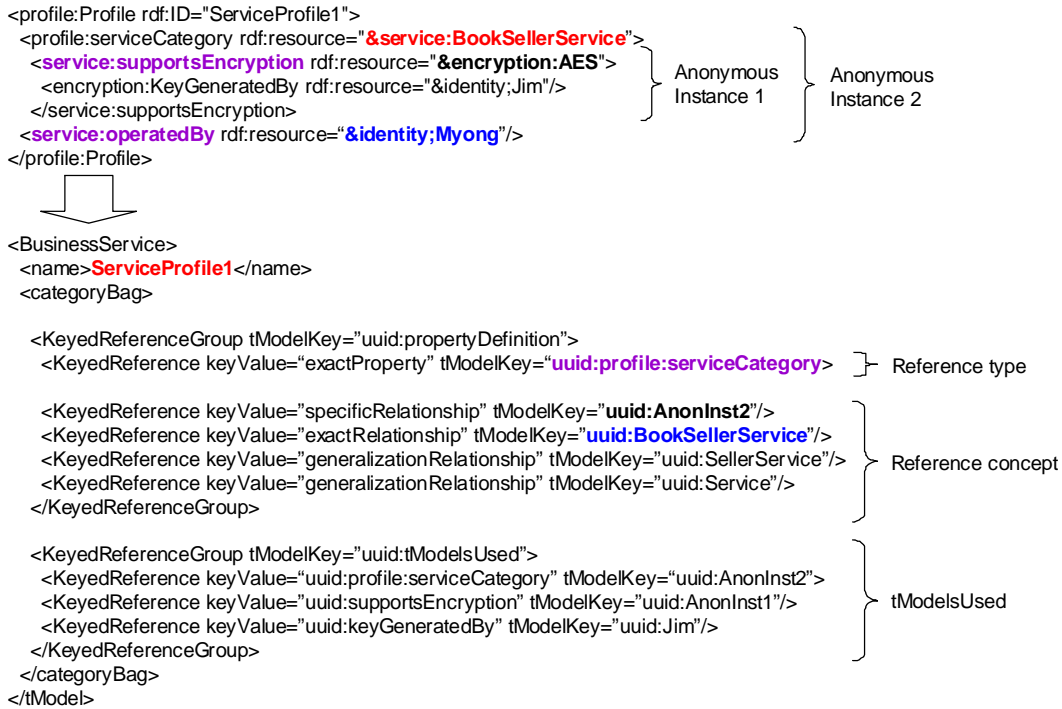

Figure 28. Translation of the service description BusinessService objects with respect to ontology concept references.

6. Matchmaking

OWL is a description language. It does not specify a query language or any algorithms for performing matchmaking. Semantic matchmakers have been developed for OWL-S [10-12]. However, these implementations are based on complex ontology reasoners or theorem provers and are incompatible with our system architecture. Our matchmaker needs to be as thin as possible and operate syntactically. Therefore, a new algorithm was developed and a new matchmaker was implemented for this prototype.

6.1. Matching Rules

Matchmaking in this system will follow the model where service providers publish capabilities and service requesters query with requirements. In order for a service description to satisfy a query, every query requirement must be matched to a service description capability. There are five possible types of match for each pair of capabilities and requirements.

1. The trivial case is a **perfect match**. That means the service description and the query specified use the exact same ontology class and property type. All the classes and properties specified are identical.
2. An **equivalent match** occurs when there exists at least an equivalence relationship between classes and properties in the service description and those in the query. There are two ways the equivalence relationship could be established
   a. The ontology explicitly states an equivalence relationship between two classes using the *equivalentClass* construct.
   b. The ontology defines a class using a class expression with the *unionOf* construct. This implies the concept is a full member of all the classes in the class expression.
3. A **specialization match** occurs when the service description is more specific than the query. A specialization match is a full match and will be treated as a match by the system. There are two ways this can occur:
   a. The requirement specifies a more general concept. For example, the service description parameter specifies AES while the query asks for Encryption. AES is a form of Encryption so it would definitely match.
   b. The requirement specifies a more general property type. For example, the service specifies operatedBy while the query asks for contactPerson.
   c. The requirement specifies a less refined concept. For example, the service description parameter specifies as BookSeller operated by Jim while the query asks for only a BookSeller.
4. A **generalization match** occurs when the service description is more general than the query. This is the opposite of a specialization match. It is only a partial match. The concept specification does not rule out a match, but the match cannot be guaranteed. This will not be treated as a match by the system. There are also two ways this can occur.
   a. The service description specifies a more general concept. For example, the service description parameter specifies Encryption while the query asks for AES. Encryption specified in the service description could be AES, but it is not certain. Therefore, this is only a partial match
   b. The requirement specifies a more specific property type. For example, the service specifies contactPerson while the query asks for ownedBy.
   c. The service description specifies a less refined concept. For example, the service description parameter specifies BookSeller while the requirement asks for BookSeller operated by Jim. The service could be a BookSeller operated by Jim, but it is not certain. Therefore, it is only a partial match.

5. A **mismatch** occurs when specification of the classes rules out the possibility of a match. There are two ways this can occur.
   a. The base class is mismatched. For example, the service description parameter specifies SellerService while the query asks for PurchasingService.
   b. The attributes are mismatches. For example, the service description parameter specifies SellerService operated by Jim while the query asks for SellerService operated by Bruce.
   c. The property type mismatch. For example, the service specifies operatedBy while the query asks for ownedBy.

The matchmaker must attempt to match every query requirement against every service capability. The degree of match for a single requirement is its highest level of match it has against all of the possible capabilities. The level of match between the requester and the service is the same as lowest degree of match for a query requirement.

- If at least one of the query requirements is not matched to a service description capability, then the query is not matched to the service description. The requestor will not be able to use the service.
- If all of the query requirements have at least a generalization match to the service description capabilities, then the there is a partial match between the query and the service description. The requester might be able to use the service, but it is not certain. It depends on the policy of the requester and the characteristics of the ontology.
- If all of the query requirements have a perfect, equivalent, or specialization match to the service description capabilities, then it is a match and the requester can indeed use the service.

Specialization matches are treated as full matches. This implies that requirements specified with a general concept will match all of the children concepts under it. For example, if the query specifies Encryption as a requirement, then all of the subclasses of Encryption in the class hierarchy as well as all the anonymous instances with additional annotations will match the query. On the other hand, generalization matches are treated as only partial matches. When a service description specifies a general concept as a capability, it does not mean the service will support all the specific concepts under the general concept. For example, if a service description specifies Encryption as a capability, it does not mean it will support all of the more specific concepts such as AES, RSA, and RSA with keys generated Jim. Therefore, it is always in the interest of the service provider to describe their service with as much detail and specificity as possible. Publishing additional details will never prevent a match that would otherwise be made. Queries, on the other hand, should be as general as possible in order to find the most matching service descriptions. Using more general concepts in a query will tend to yield more results.

6.2. Ontology Reasoner

The task of the ontology reasoner is to resolve all the identity concepts and inferred properties during the publication of the ontology.

Identity Concepts

Resolution of identity concepts of classes, instances, and proper types is the most important element in providing semantic query processing. This information will be used by both the UDDI search engine and the client-side matchmaker to perform semantic matching. The list of identity concepts depends on the matchmaking rules as well as the semantics of individual ontologies. There are four types of identity relationships based on class hierarchy that must be considered by the ontology reasoner. The relationship type is defined from the perspective of identity concept. The definition of identity concept is related concepts queries for which should yield the original concept. For example, if queries for Encryption should return AES, then class Encryption would be an identity class of AES. AES would hold a reference to Encryption as an identity concept and the relationship type of that reference is the relationship type from AES to Encryption.

The following are the possible types of identity concepts for classes and instances. They are summarized graphically in Figure 29.

1. ExactRelationship – All ontology concepts will make a reference to itself as an identity relationship.
2. EquivalenceRelationship – For a given concept, all the other classes and instances to which the ontology established an *equivalentClass* relationship. In addition, equivalence relationships are established when classes or instances is defined using the *unionOf* construct.
3. SpecializationRelationship – For a given concept, all the other classes and instances to which the ontology established a *subClassOf* relationship. Transitivity applies so that the parent class of a parent class is also a specializationRelationship identity concept.
4. GeneralizationRelationship – For a given concept, all the other classes and instances from which the ontology established a *subClassOf* relationship. Transitivity applies so that the children of a child class are also generalizationRelationship identity concepts.
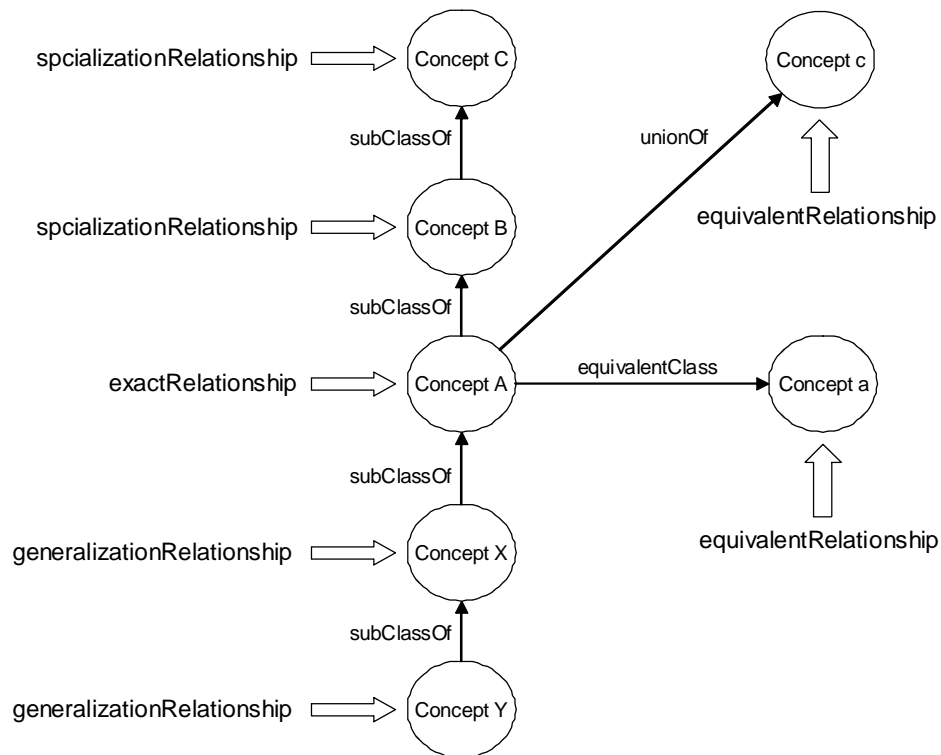
Figure 29. Identity concept derivation for classes and instances.

The following are the possible types of identity concepts for property types. They summarized graphically in Figure 30.

1. ExactProperty – all ontology concepts will make a reference to themselves as an identity relationship.
2. EquivalentProperty – all the other properties to which the property the ontology established an *equivalentProperty* relationship.
3. SpecializationProperty – all the other properties to which the ontology established a *subPropertyOf* relationship from the property. Transitivity applies so that the parent property of a parent property is a specializationProperty identity concept.
4. GeneralizationProperty – all the other properties from which the ontology established a *subPropertyOf* relationship to the property. Transitivity applies so that the children of a child property are generalizationProperty identity concepts.
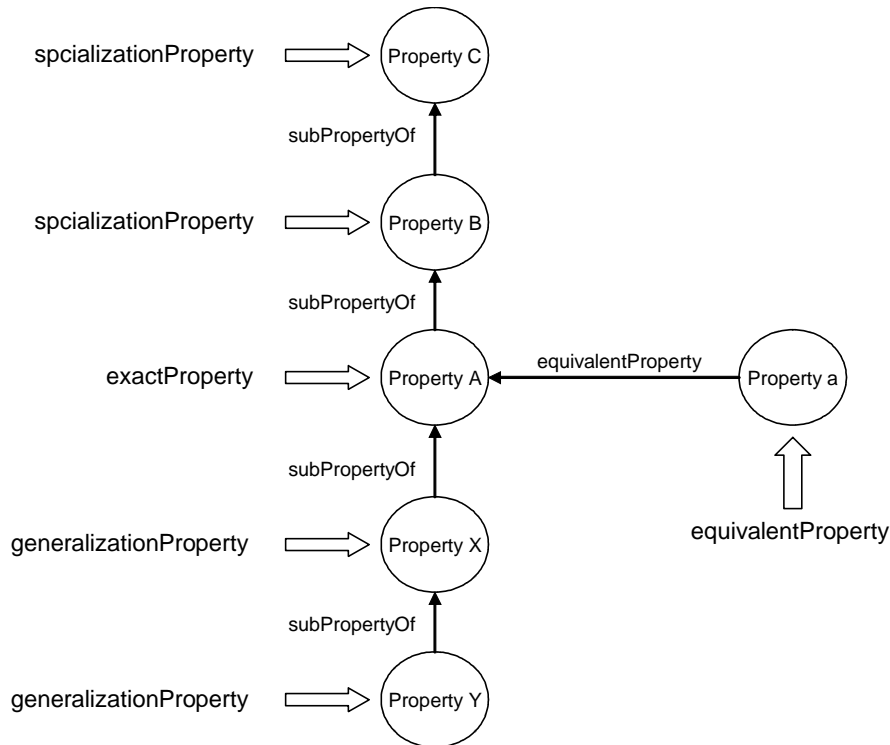
Figure 30. Identity concept derivation for property types.

The ontology reasoner can be parameterized with respect to the number of layers to apply transitivity for specialization and generalization identity concept. The publisher of the ontology can decide what makes sense for a particular ontology and implement their own modules for resolving identity concepts.

Inferred Properties

Inferred properties based on symmetric and transitive property characteristics need to be processed so that they are resolved and indexed during the publication of the ontology.

For symmetric properties, the reverse of the explicit property definition must also be defined as a separate inferred property on the property value class or instance. For example, if the coworker property is a symmetric property and Myong is defined to have the sibling Jim, then Jim needs to define the inferred property to have the coworker Myong. A new property annotation keyedReferenceGroup is created on the Jim instance for the inferred property as shown in Figure 31.
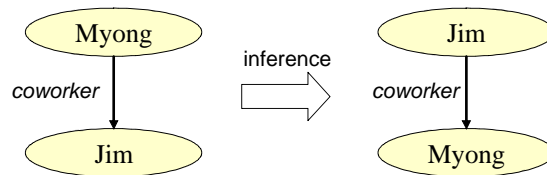

Figure 31. Inferred property based on the symmetric property characteristic

For transitive properties, the entire chain of transitivity must be resolved and referenced as additional property values in the same annotation keyedReferenceGroup. Since contactPerson property is a transitive property, for example, if Myong is defined to have the contactPerson Jim, and Jim is defined to have the conactPerson Bruce, then Myong need to reference both Jim and Bruce as his contactPerson as shown in Figure 32. They are referenced in the same property annotation keyedReferenceGroup in parallel keyedReferences. Unlike inferred properties based on symmetric properties, new property annotation keyedReferenceGroups are not created.
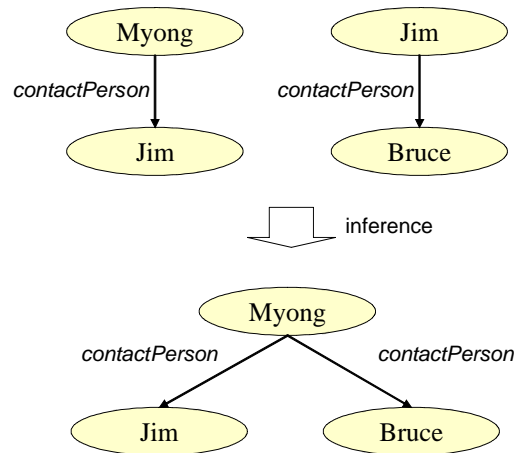

Figure 32. Inferred property based on the transitive property characteristic

When publishing service descriptions and anonymous instances, special operations need to be performed in order to propagate the chain of transitivity for transitive properties. For symmetric properties, the inferred properties can be treated exactly like explicit property annotations and processed normally without special treatment. Transitive properties, on the other hand, could require additional references in the property annotations depending on the property annotations of the property value concept. For example, if a service description is defined to have the contactPerson of Myong, then it needs to also reference Jim and Bruce. The additional property values are not identity concepts of Myong, but rather property values in one of the property annotations of Myong. If a service description or anonymous instance has a transitive property and the property value concept has same property annotations, then additional property value reference must be made to capture the entire chain of transitive properties. These operations will be performed by the publication module for service descriptions and anonymous instances.

6.3. UDDI Query Procedure

Ontology Query

Queries for the ontology take place in two stages. The first stage involves querying for the ontology overview tModel using the ontology name or URL of the external resource.

```
<find_tModel xmlns="urn:uddi-org:api_v3" >
  <name>service</name>
  <categoryBag>
   <keyedReference keyValue="ontologyReference" tModelKey="uddi:isOntologyBase">
  </categoryBag>
</find_tModel>
```
Figure 33. Query for the ontology overview tModel

The second stage then queries for all the tModels that make a reference to the ontology overview tModel using the keyValue of "ontologyReference". To locate only base ontology tModels, the query should include an additional keyedReference to the "isOntologyCore" tModel with the keyValue of "true" as shown in Figure 34. To locate only anonymous instance tModels, the query should include a keyedReference to the "isOntologyCore" tModel with the keyValue of "false".

```
<find_tModel xmlns="urn:uddi-org:api_v3" >
  <categoryBag>
   <keyedReference keyValue="ontologyReference" tModelKey="uddi:service">
   <keyedReference keyValue="true" tModelKey="uddi:isOntologyBase">
  </categoryBag>
</find_tModel>
```
Figure 34. Query for all the base ontology tModels

Ontology Base Concept TModel Query

The base ontology concepts can be queried for using their class, instance, or property names from the OWL documents as shown on Figure 35. The ontology tModel should also be included in the query because the same name might be used for distinct concepts from different ontologies.

```
<find_tModel xmlns="urn:uddi-org:api_v3" >
  <name>BookSellerService</name>
  <categoryBag>
   <keyedReference keyValue="ontologyReference" tModelKey="uddi:service">
  </categoryBag>
</find_tModel>
```
Figure 35. Query for ontology concept tModel

Complete Service Description Query

Queries for service descriptions made to the UDDI registry take place in two stages. The first stage involves retrieving from the registry all the tModel keys that are required by the second stage. A client-side query processor will query the UDDI registry for tModels of all the concepts involved in the query such as classes, instances, and property types as well as the ontology and book keeping tModels. After the keys for ontology concepts tModels are found, they can be used to construct BusinessService queries for the second stage as shown in Figure 36. The UDDI query will only deal with base class of composite concepts. The client-side matchmaker will then refine the results by matching the entire concept.

```
<find_business>
  <categoryBag>
    <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <KeyedReference tModelKey="referenceTypeKey1"/>
      <KeyedReference tModelKey="referenceValueKey1"/>
    </KeyedReferenceGroup>
    <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <KeyedReference tModelKey="referenceTypeKey2"/>
      <KeyedReference tModelKey="referenceValueKey2"/>
    </KeyedReferenceGroup>
  </categoryBag>
</find_business>
```

Figure 36. BusinessService query

The BusinessService objects will make references to identity concepts for both the property type and the property value. Queries for any of the identity concepts will return the BusinessService object. Therefore, the results of the query are effectively semantic. The requester can control the semantic matching by specifying the identity concept relationship type stored in the keyValue field of ontology concept keyedReferences. To find only a specific type of match, query with the relationship type specified in the keyValue field as shown in Figure 37.

```
<find_service xmlns="urn:uddi-org:api_v3" >
  <categoryBag>
    <keyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <keyedReference tModelKey="uddi:BookBuyerService" keyValue="exactRelationship"/>
      <keyedReference tModelKey="uddi:serviceCategory" keyValue="exactProperty"/>
    </keyedReferenceGroup>
  </categoryBag>
</find_service>
```

Figure 37. UDDI query for specific type of matches.

To find all types of matches, use the approximate match find qualifier and query with a wildcard in the keyValue field as shown in Figure 38.

```
<find_service xmlns="urn:uddi-org:api_v3" >
  <findQualifier>uddi-org:approximateMatch:SQL99</findQualifier>
  <categoryBag>
    <keyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <keyedReference tModelKey="uddi:BookBuyerService" keyValue="%"/>
      <keyedReference tModelKey="uddi:serviceCategory" keyValue="%"/>
    </keyedReferenceGroup>
  </categoryBag>
</find_service>
```

Figure 38. UDDI query for all types of matches

Partial Service Description Queries

Queries for partial service descriptions also take place in two stages. The first stage involves queries for the tModel keys of the property type and property value tModels. The second stage will query for BusinessService objects that reference those concepts under the "tModelsUsed" keyedReferenceGroup as shown on Figure 39.

```
<find_service xmlns="urn:uddi-org:api_v3" >
  <categoryBag>
   <keyedReferenceGroup tModelKey = "tModelsUsed">
    <keyedReference keyValue="uuid:keyGeneratedBy" tModelKey="uuid:Jim"/>
   </keyedReferenceGroup>
  </categoryBag>
 </find_Service>
```

Figure 39. UDDI query for partial service description queries.

The matching rules for keyedReferenceGroup require the child keyedReferences specified in the query to be a subset of the child keyedReferences in the service descriptions. To perform a global AND query, form the query with all the requirement concepts placed in the same tModelsUsed keyedReferenceGroup as shown in Figure 40. To perform a global OR query, form the query with separate keyedReferenceGroups for each requirement concept as shown in Figure 41. However, queries with complex Boolean combinations are not supported because they require Boolean expressions that simply cannot be expressed in UDDI.

```
<find_business>
  <categoryBag>
   <KeyedReferenceGroup tModelKey="tModelsUsed">
    <KeyedReference keyValue="propertyTypeKey1" tModelKey="propertyValueKey1">
    <KeyedReference keyValue="propertyTypeKey2" tModelKey="propertyValueKey2">
    <KeyedReference keyValue="propertyTypeKey3" tModelKey="propertyValueKey3">
   </KeyedReferenceGroup>
  </categoryBag>
 </find_business>
```

Figure 40. TModelsUsed query using the AND operator to combine the requirements

```
<find_business>
  <categoryBag>
   <KeyedReferenceGroup tModelKey="tModelsUsed">
 <KeyedReference keyValue="propertyTypeKey1" tModelKey="propertyValueKey1">
   </KeyedReferenceGroup>
   <KeyedReferenceGroup tModelKey="tModelsUsed">
 <KeyedReference keyValue="propertyTypeKey2" tModelKey="propertyValueKey2">
   </KeyedReferenceGroup>
   <KeyedReferenceGroup tModelKey="tModelsUsed">
 <KeyedReference keyValue="propertyTypeKey3" tModelKey="propertyValueKey3">
   </KeyedReferenceGroup>
  </categoryBag>
 </find_business>
```

Figure 41. TModelsUsed query using the OR operator to combine the requirements

As mentioned previously, the identity concepts are not referenced in the tModelsUsed keyedReferenceGroups. As a result, this category of queries will have very limited semantic capabilities. Identity concept can be derived from the concept tModels and applied to the query side. For example, queries for AES can include Encryption and SymEnc combined with the OR operator. However, the OR operator can only be applied globally and this approach will work only if there is a single requirement.

Anonymous Instance tModel Query

Anonymous instance tModels can be queried for using the exact same approach as the query for BusinessServices. The difference is that the query will be looking for tModels and the ontology overview tModel should be used. The UDDI query will use the base class of the

anonymous instance and the client-side matchmaker will match the annotations as shown in Figure 42.

```
<find_tModel>
  <categoryBag>
    <KeyedReference tModelKey = "uuid:service"/>
    <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <KeyedReference tModelKey="propertyTypeKey1"/>
      <KeyedReference tModelKey="propertyValueKey1"/>
    </KeyedReferenceGroup>
    <KeyedReferenceGroup tModelKey="uuid:propertyDefinition">
      <KeyedReference tModelKey="propertyType2"/>
      <KeyedReference tModelKey="propertyValue2"/>
    </KeyedReferenceGroup>
  </categoryBag>
</find_tModel>
```

Figure 42. Anonymous instance tModel query

6.4. Client-side Matchmaker

The UDDI search engine is capable of fulfilling by itself the first and forth categories of queries, the ontology query and the partial service description query, described in section 4. The client-side matchmaker is responsible for refining the results of the second and third categories of queries, the ontology concept query and the complete service description query. For those queries, results from the UDDI search engine only take into account the base ontology classes of composite concepts. The client-side matchmaker will fully resolve the composite concepts in the service descriptions and determine the actual level of match.

Resolution of composite concepts referenced by service descriptions is the first task of the client-side matchmaker. BusinessService objects hold annotations as keyedReferenceGroups where children keyedReferences store the keys of the concept tModels. The matchmaker in turn queries the UDDI registry for the concept tModels using the key and derives information regarding the base class and identity concepts. If the concept tModel has annotations of its own, they are resolved the same way. This process is repeated until the lowest layer concept tModels, tModels that do not have any annotations of their own, are reached. This way, the service description can be reconstructed with all the annotations and identity concepts included as shown in Figure 43.
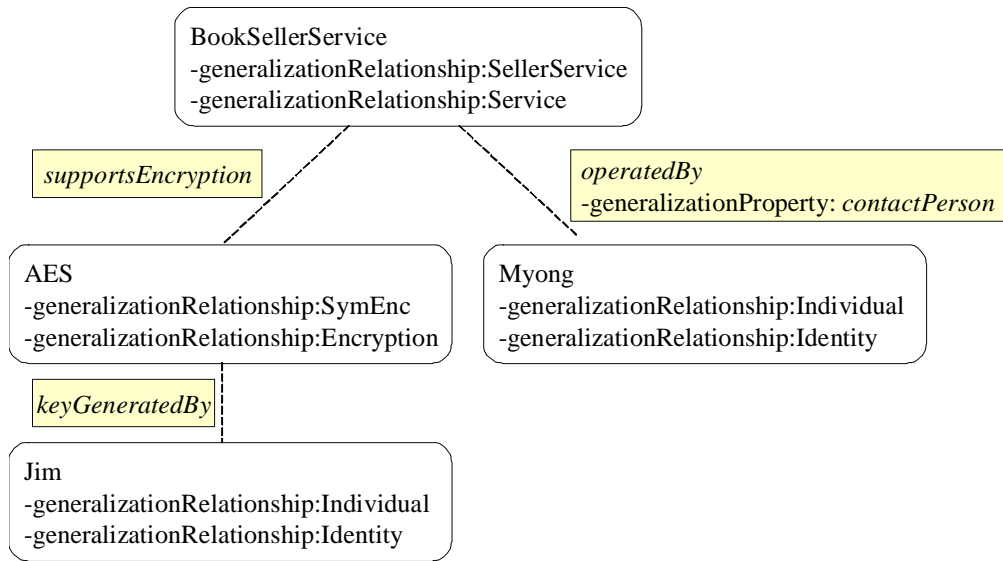
Figure 43. Service description with all concept and identity concepts fully resolved.

The second task of the client-side matchmaker is to determine the actual level of match between the query and service descriptions. This is done by matching the query concept (Figure 44) and service description concept (Figure 43) at corresponding annotation. The level of match for each individual concept can be derived from the identity relationship type of the matching concept in the reconstructed service description.
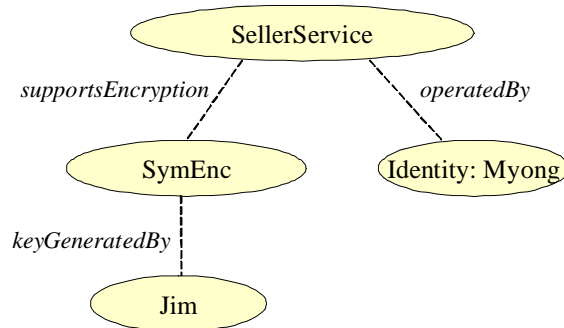

Figure 44. Query concept

7. Suggested Changes to the UDDI Standard

The OASIS group has already started work on version 4 of the UDDI specification. They are planning to provide additional support for Boolean queries and semantic taxonomies. From the perspective of this project, the existing UDDI data model is fully capable of storing semantic service descriptions. Furthermore, support for Boolean queries in the UDDI specification will allow for semantic query processing to take place completely inside UDDI registries. An important characteristic of our approach is that the modules for publishing and querying service descriptions are completely syntactic. The ontology reasoner is only used in the module for publishing the ontology itself. After the ontology is published into the registry,

service descriptions and queries that reference the ontology are processed syntactically. This means the ontology reasoner is only needed for special clients responsible for publishing ontologies. Normal users of the registry will not need any ontology aware reasoner to take advantage of semantic matchmaking.

The client-side matchmaker is only necessary due to the lack of support for Boolean queries in UDDI. It is completely syntactic and does not need to be ontology aware. The client-side matchmaker will no longer be necessary if the new version of the UDDI specification provides support for complex nested Boolean expressions such as

((A or B) AND (C or D or E) AND ((X or Y or Z) AND (U or V))

The Boolean query support must be provided for the overall category bag as well as inside keyedReferenceGroups. All matching can be folded into the registry to be performed by the UDDI search engine if complex Boolean query expression is allowed. The current translation scheme will be completely forward compatible. The same data structures can be used to support the new matching scheme.

Querying will take place in two phases. The first phase involves finding the relevant top level ontology concept tModels in the registry. In cases where the query does not contain references to composite concepts, the relevant ontology concept tModels are simply identity concepts of the query concept. First, query for the concept tModel using the name of the class as shown in Figure 45.

```
<find_tModel xmlns="urn:uddi-org:api_v3" >
  <name>Myong</name>
</find_tModel>
```
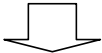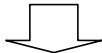
uuid:Myong

Figure 45. Query for specific base ontology concepts tModels.

Then query for the identity concepts using the tModel of the base concept as shown in Figure 47.

```
<find_tModel xmlns="urn:uddi-org:api_v3" >
  <findQualifier>uddi-org:approximateMatch:SQL99</findQualifier>
  <categoryBag>
    <keyedReference tModelKey="uddi:Myong" keyValue="%"/>
  </categoryBag>
</find_tModel>
```

uuid:Myong
uuid:Individual
uuid:Identity
uuid:AnonymousInstanceX

Figure 46. Query for simple semantically matching concept tModels

Each query could return multiple concept tModels including both base ontology concepts and anonymous instances.

If the query profile does contain composite, multiple tModel queries must be performed in a bottom up manner to find the relevant top level concept tModel. First, query for the identity concepts of the bottom-most query concept. Then query for the identity concepts of the second bottom-most query concept using the corresponding query concept as the base class and the results from the first query as property definitions. The following query in Figure 47will return tModels that semantically match the concept "BookSellerService operatedBy Myong". It will use results from the previous round of queries.

```
<find_tModel xmlns="urn:uddi-org:api_v3" >
  <findQualifier>uddi-org:approximateMatch:SQL99</findQualifier>
  <categoryBag>
   <AND>
     <keyedReference tModeKey="uddi:BookSellerService" keyValue="%"/>    ⎤  base-class from
     <keyedReferenceGroup tModelKey = "propertyDefinition">              ⎦  previous round of queries
      <AND>
        <OR>
          <keyedReference tModelKey="uuid:operatedBy" keyValue=%/>       ⎤ property type from
          <keyedReference tModelKey="uuid:contactPerson" keyValue=%/>    ⎦ previous round of queries
        </OR>
        <OR>
          <keyedReference tModelKey="uuid:Myong" keyValue=%/>            ⎤
          <keyedReference tModelKey="uuid:Individual" keyValue=%/>       ⎥ property value from
          <keyedReference tModelKey="uuid:Identity" keyValue=%/>         ⎥ previous round of queries
          <keyedReference tModelKey="uuid:AnonymousInstanceX" keyValue=%/> ⎦
        </OR>
      </AND>
     </keyedReferenceGroup>
   </AND>
  </categoryBag>
</find_tModel>

          ⬇

    uuid:AnonymousInstanceA
    uuid:AnonymousInstanceB
    uuid:AnonymousInstanceC
```

Figure 47. Query for composite semantically matching concept tModels.

Repeat this process until the relevant top level tModels are found. After that, the services can be located in the second phase by querying for BusinessService objects that reference those top level concept tModels as shown in Figure 48.

```
<find_service xmlns="urn:uddi-org:api_v3" >
  <findQualifier>uddi-org:approximateMatch:SQL99</findQualifier>
  <categoryBag>
    <AND>
      <keyedReferenceGroup tModelKey = "propertyDefinition">
        <AND>
          <keyedReference tModelKey="uuid:serviceCategory" keyValue=%/>   ⎤_  property type from
          <OR>                                                            ⎦   previous round of queries
            <keyedReference tModelKey="uuid:AnonymousInstanceA" keyValue=%/>   ⎫
            <keyedReference tModelKey="uuid:AnonymousInstanceB" keyValue=%/>   ⎬  property value from
            <keyedReference tModelKey="uuid:AnonymousInstanceC" keyValue=%/>   ⎭  previous round of queries
          </OR>
        </AND>
      </keyedReferenceGroup>
    </AND>
  </categoryBag>
</find_Service>
```

Figure 48. Query for BusinessService objects.

Since identity concepts are used in each query, the results will be effectively semantic. This way, all the work of matchmaking will be done within the UDDI registry. The client-side modules will still be necessary. The ontology reasoner will be involved in publication of the ontology. However, for publishing and querying of service descriptions, the only task for the client-side modules would be to organized multistage UDDI API invocations.

8. Related Work

Many previous attempts have been made to integrate OWL and OWL-S with UDDI registries [13-16]. There are several major differences between the previous attempts and our approach. First, our approach supports annotations of ontology concepts and composite concepts with multiple layers of annotations. Properties can be defined for classes and instances in service descriptions to create novel concepts that do not exist in the ontology. Second, our approach makes use of existing UDDI registries without extension or modification to the infrastructure.

Srinivasan et al. [13], Moreau et al. [16], and Paolucci et al [15] added semantic support to UDDI by extending the specification and placing add-on modules on the registry side. This means the existing registry infrastructure needs to be modified extensively to provide semantic support. Furthermore, the add-on modules create special interfaces and query engines for processing semantic publications and queries separate from the UDDI interface. In effect, these modules act as separate semantic registries that happen to be on the same server as opposed to truly integrating with the UDDI registry.

Sivashanmugam et al. [14] developed a scheme to store ontology-based semantic markups using the native UDDI data model. However, their solutions do not support composite concepts with multiple layers of annotations. Ontology concepts can be referenced as is, but they cannot be further annotated by service descriptions. Furthermore, class hierarchy between concepts in the ontology is not captured by the translation. It is not clear if semantic queries can be supported using this approach.

The Web Service Modeling Ontology (WSMO) based Web Service Modeling Language (WSML) is an alternative to OWL-S and OWL for semantically describing web services [8].

43

Since a direct mapping exists between WSML and OWL [9], WSMO will be supported indirectly in our system.

9. Conclusion

We presented an approach for supporting semantic markups of Web services and semantic queries using existing registries conforming to the UDDI V3 specification. Support is provided for the OWL language as a whole and the system will operated with any OWL ontology including OWL-S. A special lossless translation scheme that fully supports composite concepts was developed to store ontologies and semantic service descriptions inside UDDI registries. Once all the semantic information is captured, semantic query processing can be performed using a combination of the UDDI search engine and syntax based client-side matchmaker.

This approach does not require any modification to the existing registry or infrastructure. The advantage is that it is completely backward compatible. The add-on modules only need to be installed on the clients of users who wish to take advantage of semantic markups. They can be integrated seamlessly into existing systems and operations without any modification of the infrastructure

References

1. W3C Recommendation, "OWL Web Ontology Language Guide," W3C. 2004 < http://www.w3.org/TR/owl-guide/ >.
2. Web Ontology Working Group, "OWL-S: Semantic Markup for Web Services," W3C. <http://www.daml.org/services/owl-s/1.1/overview/>.
3. OASIS, "UDDI Executive Overview: Enabling Service-Oriented Architecture," OASIS. 2004 <http://uddi.org/pubs/uddi-exec-wp.pdf>.
4. A. Dogac, G. Laleci, Y. Kabak, and I. Cingil, "Expoliting Web Service Semantics: Taxonomies vs. Ontologies," *IEEE Data Engineering Bulletin*, vol. 25, 2002.
5. D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. D. McDermott, McGuinness, B. Parsia, T. R. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara, "Bringing Semantics to Web Services: The OWL-S Approach," presented at First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, California, USA., 2004.
6. UDDI Spec Technical Committee, "UDDI Version 3.0.2," OASIS. 2004 <http://uddi.org/pubs/uddi_v3.htm>.
7. W3C, "Web Services Description Language (WSDL) 1.1," W3C. 2001 <http://www.w3.org/TR/wsdl>.
8. "UNSPSC Homepage," UNSPSC. 2005 <http://www.unspsc.org/>.
9. W3C, "OWL Web Ontology Language Reference," http://www.w3.org/TR/owl-ref/. 2004
10. M. Jaeger and S. Tang, "Ranked Matching for Service Descriptions using DAML-S," presented at Enterprise Modelling and Ontologies for Interoperability (EMOI), INTEROP 2004, Riga, Latvia, 2004.
11. G. Denker, S. Nguyen, and A. Ton, "OWL-S Semantics of Security Web Services: a Case Study," presented at 1st European Semantic Web Symposium, Heraklion, Greece, 2004.
12. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," presented at First International Semantic Web Conference (ISWC2002), 2002.
13. M. P. N. Srinivasan, and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput," presented at First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, California, USA, 2004.
14. K. V. K. Sivashanmugam, A. Sheth, and J. Miller, "Adding Semantics to Web Services Standards," presented at International Conference on Web Services, 2003.
15. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Importing the Semantic Web in UDDI," presented at E-Services and the Semantic Web (ESSW02), 2002.
16. L. Moreau, S. Miles, J. Papay, K. Decker, and T. Payne, "Importing the Semantic Web in UDDI," presented at Web Services, E-Business and Semantic Web Workshop, 2002.

Appendix A. OWL Representation of Example Ontologies from Figure 1.

A1. Service Ontology (service ontology.owl)

```
<!DOCTYPE rdf:RDF [
  <!ENTITY service     "http://localhost/service ontology.owl">
  <!ENTITY encryption  "http://localhost/encryption ontology.owl">
  <!ENTITY identity    "http://localhost/identity ontology.owl">
  <!ENTITY identity    "http://localhost/identity ontology.owl">
  <!ENTITY rdf         "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs        "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl         "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd         "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY dsig        "http://www.w3.org/2000/09/xmldsig#">
  <!ENTITY time        "http://www.isi.edu/~pan/damltime/time-entry.owl#">
]>

<owl:Ontology>
 <rdfs:comment>
   A service ontology
 </rdfs:comment>
</owl:Ontology>

<owl:Class rdf:ID="Service">
 <rdfs:label>Service</rdfs:label>
 <rdfs:comment>
   Base class for services
 </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="PurchasingService">
 <rdfs:comment>
   Purchasing service
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="#Service"/>
</owl:Class>

<owl:Class rdf:ID="SellerService">
 <rdfs:comment>
   Selling service
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="#Service"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="supportsEncryption">
 <rdfs:domain rdf:resource="#Service"/>
 <rdfs:range rdf:resource="&encryption;Encryption"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="contactPerson">
 <rdf:type rdf:resource="&owl;TransitiveProperty"/>
```

```
  <rdfs:domain>
   <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#Service"/>
     <owl:Class rdf:about="&identity;Identity"/>
        </owl:unionOf>
   </owl:class>
 <rdfs:range rdf:resource="&identity;Identity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="operatedBy">
 <rdfs:subPropertyOf rdf:resource="#contactPerson"/>
 <rdfs:domain rdf:resource="#Service"/>
 <rdfs:range rdf:resource="&identity;Identity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ownedBy">
 <rdfs:subPropertyOf rdf:resource="#contactPerson"/>
 <rdfs:domain rdf:resource="#Service"/>
 <rdfs:range rdf:resource="&identity;Identity"/>
</owl:ObjectProperty>

</rdf:RDF>
```

A.2 Encryption Ontology (encryption ontology.owl)

```
<!DOCTYPE rdf:RDF [
  <!ENTITY encryption  "http://localhost/encryption ontology.owl#">
  <!ENTITY identity    "http://localhost/identity ontology.owl#">
  <!ENTITY rdf         "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs        "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl         "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd         "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY dsig        "http://www.w3.org/2000/09/xmldsig#">
  <!ENTITY time        "http://www.isi.edu/~pan/damltime/time-entry.owl#">
  <!ENTITY identity    "http://localhost/identity ontology.owl#">
]>

<owl:Ontology>
  <rdfs:comment>
    A encryption ontology
  </rdfs:comment>
</owl:Ontology>

<owl:Class rdf:ID="Encryption">
  <rdfs:label>Encryption</rdfs:label>
  <rdfs:comment>
    Base class for encryption algorithms
  </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="SymEnc">
  <rdfs:comment>
    Symmetric encryption
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Encryption"/>
</owl:Class>

<owl:Class rdf:ID="AES">
  <rdfs:comment>
    AES encryption
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#SymEnc"/>
</owl:Class>

<owl:Class rdf:ID="DES">
  <rdfs:comment>
    DES encryption
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#SymEnc"/>
</owl:Class>

<owl:Class rdf:ID="AsymEnc">
  <rdfs:comment>
    Asymmetric encryption
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Encryption"/>
</owl:Class>
```

```
<owl:Class rdf:ID="RSA">
  <rdfs:comment>
    RSA encryption
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#AsymEnc"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="KeyGeneratedBy">
  <rdfs:domain rdf:resource="#Encryption"/>
  <rdfs:range rdf:resource="&identity;Identity"/>
</owl:ObjectProperty>

</rdf:RDF>
```

A.3 Identity Ontology

```
<!DOCTYPE rdf:RDF [
  <!ENTITY identity      "http://localhost/identity ontology.owl">
  <!ENTITY rdf           "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs          "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl           "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd           "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY dsig          "http://www.w3.org/2000/09/xmldsig#">
  <!ENTITY time          "http://www.isi.edu/~pan/damltime/time-entry.owl#">
]>

<owl:Ontology>
  <rdfs:comment>
    A identity ontology
  </rdfs:comment>
</owl:Ontology>

<owl:Class rdf:ID="identity">
  <rdfs:label>identity</rdfs:label>
  <rdfs:comment>
    Base class for identities
  </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Jim">
  <rdfs:comment>
    Jim's identity
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#identity"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#contactPerson" />
      <owl:hasValue rdf:resource="#Bruce"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Myong">
  <rdfs:comment>
    Myong's identity
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#identity"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#contactPerson" />
      <owl:hasValue rdf:resource="#Jim"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Bruce">
  <rdfs:comment>
    Bruce's identity
```

```
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#identity"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#contactPerson" />
        <owl:hasValue rdf:resource="#Anya"/>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>

</rdf:RDF>
```

Appendix B. OWL Representation of Example Service Description from Figure 2.

```
<!DOCTYPE rdf:RDF [
  <!ENTITY service      "http://localhost/serviceontology.owl">
  <!ENTITY encryption   "http://localhost/encryption ontology.owl">
  <!ENTITY identity     "http://localhost/identityontology.owl">
  <!ENTITY rdf          "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs         "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl          "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd          "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY dsig         "http://www.w3.org/2000/09/xmldsig#">
  <!ENTITY time         "http://www.isi.edu/~pan/damltime/time-entry.owl#">
]>

<profile:Profile rdf:ID="ServiceProfile1">
<profile:ServiceName>Web Service 1</profile:ServiceName>
<profile:textDescription>A web service</profile:textDescription>
<profile:serviceCategory rdf:resource="#ServiceParam1"/>
</profile:Profile>

<service:BookSellerService rdf:ID="ServiceParam1">
        <service:supportsEncryption rdf:resource="&encryption:AES">
                <encryption:KeyGeneratedBy rdf:resource="&identity;Jim"/>
        </service:supportsEncryption>
        <service:operatedBy rdf:resource="&identity;Myong"/>
</ service:BookSellerService>
```